Universitat
Autònoma
de Barcelona

Exploiting the structure of Distributed Constraint Optimization Problems to assess and bound coordinated actions in Multi-Agent Systems

A dissertation submitted by
Meritxell Vinyals Salgado
at Universitat Autònoma de Barcelona to
fulfill the degree of **PhD** in Computer Science.

Bellaterra, March 28, 2011

Director: **Dr. Juan Antonio Rodríguez-Aguilar**
Institut d'Investigació en Intel·ligència Artificial
Consell Superior dInvestigacions Científiques

Co-director: **Dr. Jesús Cerquides Bueno**
Institut d'Investigació en Intel·ligència Artificial
Consell Superior dInvestigacions Científiques

Pots assolir aquell cim sense basarda
si vols pots dur la barca mar endins,
gaudir del teu jardí quan cau la tarda,
de la tela basta fer mocadors fins,

Allunye'n de tu Meri, l'altivesa,
de les teves virtuds fes-ne un pomell,
tot l'entorn adornat amb senzillesa,
farà que el teu jardí sigui el més bell.

L'avi Francesc

Agraïments

Aquesta tesis, més que el resultat del meu treball, ha estat resultat de la meva interacció amb tot un conjunt de persones sense les quals res hagués estat igual. A arribat l'hora doncs de treure aquestes persones de l'anònimat i donar-los les gràcies i el reconeixement que es mereixen.

Primer de tot dono les gràcies als meus directors, en Jar i el Jesús, els quals han estat autors actius de tot aquest treball. Al Jar, per no tindre mai un no per resposta, per posar un to aristocràtic anglès en tots els meus articles (amb en les infinites correccions que això ha comportat) i, sobretot, per encomanar-me el seu inesgotable optimisme. Al Jesús, per totes les idees brillants, esforç, dedicació i comentaris sincers que han donat contingut i formalisme a aquesta tesis. Als dos, per no haver-me deixat sola en aquest llarg camí, ni tan sols quan el vaig girar per deixar enrera les MMUCA.

Aquesta tesis s'ha portat a terme en un lloc molt especial, el IIIA, el qual ha estat com la meva segona casa durant els últims quatre anys. A tota la gent del centre, des dels doctors fins a les noies d'administració, m'agradaria donar-los les gràcies per haver-me fet sentir com a casa durant tot aquest temps. Al Dani Polak, que em va obrir la porta de l'institut quan hi vaig aterrar amb una beca patinet i me l'ha seguit obrint durant tots aquests anys. A l'Isa i Ana que han sabut conviure amb el meu desordre. A Tito, per ser tant *superadmin*. També m'agradaria agraïr especialment a en Carles, ja que després de parlar amb ell vaig decidir quedar-me a fer la tesis , per fer-se pròxim amb la seva energia i veu de tro i per no tindre mai un no per resposta sempre que he necessitat un cop de mà. Finalment, aquest doctorat no hagués arribat a la seva meta sense uns companys que, a més a més d'omplir tots aquests anys d' experiències inolvidables, no m'han deixat decaure en cap moment. En especial a tots aquells amb els quals he compartit més part d'aquest trajecte junts. A en Dani, *el bonico del to*, per tots els seus "guapa" matinals i el seu inesgotable optimisme i humor que han impulsat la majoria d'actes socials d'aquest grup. A l'Angi, *the google girl*, per la seva confiança i per totes les bogeries i aventures que hem compartit. A Dr. Pinyol, el hombre más bueno del mundo, per totes les converses, riures i paraules sensates. A Sr. Nin, un amic que ha estat sempre diposat a fer-te un favor mentre et contava les fatalitats del nostre futur doctoral. A Mari, que tot i la seva incorporació més tardana, ja no em sé imaginar el IIIA sense ella ni el seu constant suport. A l'Arnau, per aquells temps en la sala de robòtica, amb el projecte de final de carrera, i perquè sempre he pogut comptar amb ell durant els anys de doctorat que l'han seguit. Al Marc per a contribuir activament en aquesta tesis tant en el seu treball de final de carrera com a en la seva tesina de màster.

A JL, el único andaluz que he dejado que lo estresara sin soltar queja. A Norman, por cuidarme en nuestro primer viaje a Boston y por toda la reposteria que nos hemos comido a costa de su estrés. Al despatx 311, que sempre va ser una mica meu. A la ciència, que ens ha portat a viatjar a congressos per tot el món i per la qual hem brindat en cada ocasió .

Al Javi, Marc, i Alex pel gran esforç que van dedicar en els projectes de finals de carrera els quals vaig tenir la sort de poder co-dirigir.

To Prof. Victor Lesser and the Amherst group for kindly accepting me in the short stay I realised there during the first year of my PhD. In special to Yoonheui, Chongjie, Xiaoxi, Michele and Hala for offering me their company and friendship when I was so far from home. I would also like to thank Alan and Akshat for all the moments we shared in NIPS conference years later.

To the Santa Fe institute for offering me one of the best experiences of my PhD: an scholarship to attend one month Complex Systems Summer School in Santa Fe with amazing lectures and even more amazing participants. I would like to thank all of them for all the good memories I have from these days. In special to Ana Marín Gonzalez, Lisa Friendland, Alan Campbell and Rob Mills which I had the opportunity to meet again around the world.

Also, I would like to express my most sincerely gratitude to Prof. Nick Jennings and the IAM group for the short stay I realised in Southampton during the last year of my PhD. Many thanks to Kate, Gopal, Alessandro, George, Ruben, Francesco, Maria, Victor, Oleksandr, Rama, Alex, Simon, Archie, Sid and Krishnen. I enjoyed a wonderful experience in Southampton.

To Prof. Milind Tambe, for his interest in my work and his invaluable collaboration.

Als de la uni, al José (i Marie), Alex, JL, Santi, Manel (i Laura), Trini i Toni per tots els moments que hem seguit compartint junts i per la nova generació que esperem amb il·lusió. En especial a JL, el meu company inseparable de saf on tot l'estrès de la feina se'n anava entre converses, córrer, màquines, abdominals i lucecitas.

Al Jordi, per creure en mi, fer-me feliç, fer-me més forta i ser el meu company en tants viatges, des de Granada, la qual va tindre un gran significat per mi, fins a Southampton.

A la meva família. Als meus pares i al meu germà. Sé que no hagués estat necessari escriure una tesis perquè estiguessin orgullosos de mi. Ho han estat sempre. Ells tenen el do de fer-me sentir important en els petits èxits, recolsant-me en tots els entrebancs. És fàcil tirar endavant amb una família com aquesta al costat.

A tot a la colla d'avis que m'han mimat tots aquests anys. Als meus avis Neus i Alex que han estat per mi exemples de com viure i estimar. Als meus avis Francesc i Teresa, per haver fet voluntat de compartir i ajudar-me en tots els moments importants de la meva vida. A l'Elvira, per a la seva estima incondicional que s'ha expressat en tantes atmelles, sanfaina, verdures i un llarg etcètra sense les quals no hagues disposat de la força per acabar aquesta feina.

Per últim, vull dedicar aquesta tesis a la memòria de l'avi Francesc i a la lliçó que em va intentar ensenyar: *ser sàvia et farà important, ser estimada et farà feliç*. Encara que necessitaré més temps per apendre aquesta lliçó, sé que l'apendré, doncs són les sàvies paraules d' un home que serà recordat per tots els que vam tenir la sort d'estimar-lo.

Contents

List of Figures

Abstract

This thesis focuses on Distributed Constraint Optimization as an approach to coordinate the actions of a network of cooperative agents. Different scenarios pose very different problems if we intend to endow agents with coordinated behaviour due to availability of resources. Some resource-bounded problems require complete DCOP algorithms that can find the most effective use of resources to achieve an optimal coordination, while others need incomplete algorithms that sacrifice optimality in favour of low-cost suboptimal solutions.

This motivates the main goal of this dissertation: the design of efficient DCOP algorithms to cope with different resource-bounded scenarios while ensuring that agents select the actions that allow their coordination. Quality assessment is likely to play an important role in this endeavour, because it is fundamental to weigh the cost of coordination against the quality of the solution reached (trade-off quality versus cost). Unfortunately, quality assessment for incomplete DCOP algorithms is a major challenge that requires a broader concept of guarantees that includes approximate quality guarantees.

This thesis addresses these two major issues: efficiency and approximate quality assessment. We make significant contributions to both issues. The main idea behind these contributions is to design algorithms, and frameworks, which exploit a DCOP structure, namely the structure of agents dependencies and of their rewards, to assess and bound high quality solutions.

Regarding optimality guarantees, we contribute with Action-GDL, a complete DCOP algorithm that generalises and improves the efficiency of existing dynamic programming DCOP algorithms, unifying them under the GDL framework. The key idea behind Action-GDL is to better exploit a DCOP structure by using a novel problem representation based on junction trees.

As to approximate quality assessment, we propose two general frameworks: Divide-and-Coordinate (DaC) and Region Optimality.

First, the DaC framework enables to trade-off quality versus cost from an agent perspective by defining a family of algorithms, the DaC family, which allows agents to bound the quality of their solutions at run time. The DaC framework defines a new approach to DCOP solving by: (i) splitting the problem into simpler sub-problems that are computationally tractable; and (ii) having sup-problems agree on a solution. We define DaCSA and EU-DaC, two DaC-compliant, incomplete DCOP algorithms. Both algorithms can return anytime solutions with quality guarantees.

Secondly, this dissertation introduces the region optimal framework, a general

framework that generalises local optimality frameworks introduced in the literature such as k-size or t-size. Region optimality allows to obtain quality guarantees for arbitrary criteria and depending on the available information about a DCOP. Moreover, we show that there are criteria that outperform state-of-the-art local criteria such as (k-)size or (t-)distance by introducing the novel size-bounded distance criterion. This contribution finishes by proposing $\mathcal{C}$-DALO, an asynchronous region optimal DCOP algorithm to search for region optimal solutions in any region characterised by any arbitrary criteria.

Finally, we prove that region optimality is a valuable tool for bounding the quality of the solutions achieved by the Max-Sum algorithm on convergence. These results shed light on the relationship between the Max-Sum performance and the structure of the problem. Moreover, they help identify new classes of graph structures for which Max-Sum is guaranteed to converge to high-quality solutions.

Chapter 1

Introduction

This thesis focuses on cooperative Multi-Agent Systems (MAS) where a network of agents have to coordinate their actions in order to achieve some system performance. That is the case of many MAS scenarios such as coordination, scheduling or task allocation problems where agents need to choose individual actions whose outcomes are dependent on the actions of other agents. For example, disaster management rescue agents (i.e. emergency or police units) are required to coordinate their activities to form teams of multiple responders because no single agent has all the resources to perform a rescue task (i.e. extinguish a fire, save victims).

Among the multiple frameworks that have been proposed to handle cooperative Multi-Agent coordination, this thesis focuses on Distributed Constraint Optimization. One of the main advantages of Distributed Constraint Optimisation is its capacity to capture the locality of agents' interactions by means of a graphical model. Additionally, with respect to other Multi-Agent coordination frameworks (i.e. belief-desire-intention or distributed POMDPs), it improves the computationally tractability of the model at the cost of sacrificing the representation of some problem features (i.e. agents' cognitive capabilities or the uncertainty over the outcome of agents' actions) (Tambe et al., 2005). Hence, agents' interactions are compactly modeled in a Distributed Constraint Optimization Problem (DCOP) by means of a graphical model where variables stand for agents' actions and each edge across variables denotes a subsets of agents whose joint action incurs some cost or rewards to coordination. For example, task allocation in a disaster scenario can be effectively modeled as a DCOP by creating variables corresponding to rescue agents' actions and edges to coordinate the actions of each rescue task among all agents that may contribute to it. Each rescue task generates a reward for each possible joint configuration of rescue agents that expresses its degree of accomplishment with the allocated resources. Now, the problem is to maximise the overall utility of agents joint action determined by the sum of rewards of all these rescue tasks.

Unlike centralised approaches where the whole problem is communicated and solved by a single agent (Schiex et al., 1995), DCOP solving agents are required to coordinate in a decentralised manner, by exchanging messages with other agents in a local neighborhood, to agree on the best joint decision. This decentralisation is suitable for multiple reasons. First, it allows to preserve the autonomy and decentralised struc-

ture of problems that are decentralised by nature (e.g. information is spread across the network or or the system is composed of multiple units with limited capabilities). That is the case of rescue agents in disaster management that are physically distributed in an large area (i.e. a city, a neighborhood) with access only to the information about the rescue tasks in their surroundings. Moreover, decentralisation fosters parallelism, robustness and scalability with respect to centralised approaches. Thus, decentralisation in disaster management is motivated by robustness, to avoid a single point of failure, and scalability, because agents can only communicate locally due to severe communication restrictions.

1.1 Problem Statement

Recent research in the area of Distributed Constraint Optimisation is motivated by some challenging new applications, such as the control of environmental monitoring sensor networks (Stranders et al., 2010), the deregulation of power networks (Petcu and Faltings, 2008) or coordination in large-scale disaster management (Ramchurn et al., 2010). In these domains agents, and therefore DCOP techniques, have to cope with different limitations on resource availability. For instance, if emergency units in a disaster management environment take minutes before being assigned a task or need a very expensive controller, then such coordination is no longer useful. Along this line, DCOPs have been identified as one of the MAS key techniques to contribute to the deployment of sensor networks (Rogers et al., 2009; Vinyals et al., 2010) by autonomously coordinating sensors actions to achieve their system-wide goals. As we pointed out in (Vinyals et al., 2010), the features of a sensor network (i.e. physical hardware, scale) and its environment, along with the goals it pursues, must be carefully considered because different features lead to different constraints, and hence to different problems.

Under resource-boundedness some problems require complete DCOP algorithms that can find the most effective use of resources to achieve optimal coordination. For example, consider the use of sensor networks to enable intelligent lighting control in a building (Singhvi et al., 2005; Park et al., 2007) with two primary objectives: user comfort and energy costs. DCOPs enact the trade-off between these two objectives by allowing agents to coordinate lamp settings (i.e. intensity levels) to meet users' comfort whereas minimising their consumption. Fortunately, in this domain, agents are typically deployed in a very reliable environment, linked through wireless or wired communication channels and electric-power. Moreover, the building architectural design structure allows to break coordination into smaller zones that allow independent light control (i.e. different rooms or other architectural separators), defining coordination problems of small or medium scale that allow optimal coordination.

However, in contrast with this first scenario, other problems require for more efficient coordination due to severe resource restrictions. As an example, consider a network of sensors deployed for environmental monitoring whose sensors are required to coordinate to maximise the amount of gathered information and reduce the waste of sensing resources. Given the vast area to be covered and the possibility of damaging sensors on deployment, this kind of sensor network is typically composed of a large number of small-sized battery sensors that communicate through radio frequency

transmission. Hence, coordination in this second scenario requires incomplete DCOP algorithms that sacrifice optimality in favour of low-cost suboptimal solutions.

This lack of optimality guarantees due to scarcity of resources is a situation that DCOP techniques will need to face more and more often in the near future to cope with emerging large-scale real-world problems. It is important to guarantee that individual agents select actions that result in coordinated behaviour, although maybe not optimally. Quality assessment plays an important role in this endeavour. Because *one cannot improve what one cannot measure*, quality assessment over coordination is fundamental for agents and/or for the system designer in order to weigh the cost of coordination against the quality of the solution reached (trade-off quality versus cost).

First, from an agent perspective, agents require some measure of self-awareness over the efficiency of their actions in order to trade-off quality versus cost at runtime. For example, agents in environmental monitoring need to know if it is worth investing more resources on coordination to improve their energy consumption. Alternatively, disaster management agents are required to coordinate while avoiding the risk of catastrophic effects (in terms of human and structural factors). Hence, from an agent perspective, resource-boundedness efficient coordination poses the following fundamental questions at runtime:

(A:Q1) *Should I invest more resources on coordination to obtain a better solution?*

(A:Q2) *Is the quality of my current solution above some critical threshold?*

To provide an answer to these questions, agents need to be aware of the quality of the solutions they explore at runtime. By answering question (A:Q1) agents can intelligently trade-off quality versus cost. Alternatively, by answering question (A:Q2) agents can discard solutions that do not guarantee a minimum performance quality.

Second, from a system designer perspective, he/she must be able to evaluate design alternatives to trade-off quality versus cost at design time. In particular, at design time, the system designer needs to face decisions such as which algorithm to choose (algorithm selection) and/or how to configure the problem (configuration selection).

As to algorithm selection, the system designer aims to exploit any a-priori knowledge about the characteristics of the problem to select the algorithm that ensures better agents' coordination. For example, consider designing a network of agents to synchronize traffic lights in a city. Although the best signal plan configuration would vary with traffic conditions, the structure of such synchronisation is fixed and determined by the particular urban grid (e.g. road junctions and crossings). Therefore, the system designer can exploit this a-priori knowledge to select the most suitable algorithm that would vary from city to city and from neighborhood to neighborhood.

In contrast, for those characteristics of the problem that are configurable, the system designer aims to select a configuration that eases coordination. For example, when deploying environmental monitoring sensor network, the system designer aims to select a placement for sensors. When considering mobile sensors, he would look for a formation that in addition to support information gathering also advocates in favor of coordination.

Hence, from a system designer perspective, enabling efficient resource-bounded raises the following fundamental questions at design time:

(D:Q1) *Which algorithm leads to better coordination under the known problem characteristics?*

(D:Q2) *Which problem configuration allows better coordination?*

Both questions are of crucial importance to enable algorithm and configuration selection to trade-off quality versus cost.

This motivates the main goal of this dissertation: the design of efficient DCOP algorithms to cope with different resource-bounded scenarios while ensuring, by means of quality assessment, that agents select the actions that allow their coordination. In the next section, we analyse the challenges that these two interrelated problems, resource-boundedness and quality assessment, pose on the design of DCOP algorithms.

1.2 Challenges

This section analyses the challenges regarding the design of efficient DCOP algorithms that provide quality guarantees over their solutions in scenarios with different levels of resource-boundedness.

First, we focus on domains where the cost of coordination of an optimal solution is affordable. An example of this kind of domains is the lighting control problem in a building scenario described in section 1.1. Quality assessment in this scenario is straightforward because we can apply complete DCOP algorithms that are guaranteed to return the optimal solution. In contrast, the challenge in the design of complete DCOP algorithms consists in maximising the efficiency on the use of limited resources. The efficiency of a DCOP algorithm is measured along three dimensions:

- *Communication:* the number and size of messages exchanged by agents.
- *Computation:* required by agents.
- *Parallelism:* the number of operations that agents can execute in parallel. Due to the distributed nature of DCOPs, it is desirable that operations are equally distributed among agents, minimizing the amount of time that agents are idle on the system.

Hence, the design of efficient complete DCOP algorithms aims to minimize the computation and communication required by agents and maximise the parallelism.

	Time		**Specificity**		
	Design Time	Runtime	Problem-independent	Per-Class	Per-Instance
Agent		✓			✓
Designer	✓		✓	✓	

Table 1.1: Requirements for an agent and a system designer's approximate quality guarantees regarding time and specificity.

Secondly, we consider domains in which achieving an optimal coordination is simply unaffordable. Examples of this kind of domains range from traffic or power control to distributed task allocation in disaster management.

These domains require extremely efficient incomplete DCOP algorithms that return locally optimal solutions, but only require a small amount of computation and local communication per agent. However, quality assessment for incomplete algorithms is a major challenge, an objective of research that has attracted recently much attention in the DCOP community (Pearce and Tambe, 2007; Rogers et al., 2011; Yeoh et al., 2009; Petcu and Faltings, 2005a). This challenge requires a broader concept of guarantees that, in addition to optimality, includes approximate guarantees. Approximate quality assessment stands for assessing guarantees for solutions that may not be optimal but are guaranteed to be within a given distance from the optimal one.

As advanced in section 1.1, approximate guarantees have different requirements depending on whether the quality assessment is realised by agents or by the system designer. Concretely, here we observe that these requirements vary among two dimensions: the time when quality guarantees are available and the specificity of guarantees with respect to a particular problem. Table 1.1 depicts the different categories in these dimensions. As time availability, we classify approximate quality guarantees into two categories:

- *design time*, namely quality guarantees assessed before coordination; and
- *runtime*, namely quality guarantees assessed during coordination;

As to specificity, we can classify quality guarantees into three categories (from more specific to more general):

- *per-instance* that are dependent on the particular instance of the problem, that is apply to a particular DCOP;
- *per-class* that apply to a class of DCOPs namely those that have some problem's characteristics/structure;
- *problem-independent* that apply to any DCOP, independently of its characteristics.

Intuitively, the more the knowledge about a problem, the tighter the quality guarantees. That is the reason why per-instance quality guarantees, which only apply to a particular problem instance, are expected to be much tighter than per-class and problem-independent guarantees.

First, we analyse the requirements of the quality guarantees from an agent perspective. We shall refer to them as agent's quality guarantees. As discussed in section 1.1, to trade-off quality versus cost agents need to be aware of the quality of the solutions they explore at runtime. Hence, agent's guarantees must be assessed anytime at runtime over the quality of the solution that agents currently explore. Moreover, at runtime agents know exactly the problem they are solving, namely their dependencies with other agents and rewards for their actions. For example, in disaster management, rescue agents at any particular execution point know exactly: the set of rescue tasks they can participate to, the set of agents they need to coordinate with, and how much resources need

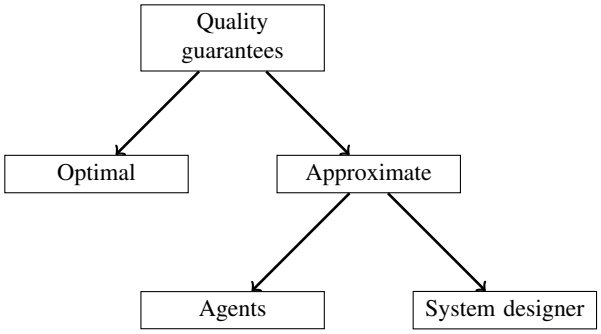


Figure 1.1: Quality guarantees classification.

to be assigned to each task to be successfully accomplished. Hence, with respect to specificity, at runtime agents aim to bound the quality of the solutions of the particular DCOP by assessing quality guarantees with the maximum specificity with respect to the problem instance, namely *per-instance* quality guarantees. Table 1.1 summarises the quality guarantees that agents require.

Second, we analyse the requirements of the quality guarantees from a system designer perspective. We shall refer to them as system designers quality guarantees. As discussed in section 1.1, system designer must assess these quality guarantees at design time over the quality of the solution that an algorithm achieves on convergence. This is a very challenging task, because at design time there is uncertainty about the specific problems that agents will deal with when deployed. For example, consider designing a DCOP algorithm for coordination in disaster management. In this domain the set of dependencies of an agent depends on their current position and on the active set of rescue tasks. Additionally, until a rescue task is not discovered, agents do not know how many rescue units are required to accomplish it. Both, agents positions and rescue tasks change over time. Therefore, system designer in these domains requires the assessment of quality guarantees that apply to any problem (*problem-independent* guarantees). That is not the case of other domains where the agent's dependencies (graph structure) or some knowledge about the rewards (reward structure) are known at design time. For example, as argued in section 1.1, agent's dependencies are fixed in traffic control or determined by sensors positions/formations in environmental monitoring. Alternatively, the system designer may know the maximum and minimum rewards of a measurement in environmental monitoring or assume rewards superadditive in disaster management (the more rescue units you add to a task the more reward you get from it). As pointed out in the section 1.1, system designer in these domains must assess *per-class* quality guarantees that use this a-priori knowledge about the characteristics of the problem. Hence, when the characteristics exploited are the structure of agents' dependencies, we will call these guarantees as *per-structure* quality guarantees. Alternatively, when the guarantees exploit some characteristics of the rewards, we call these guarantees *per-reward* quality guarantees.

Table 1.1 summarises the requirements of the system designer's approximate guarantees, namely problem-independent and per-class quality guarantees assessed at de-

sign time.

In summary, under resource-boundedness, DCOP algorithms can be classified in two categories: complete and incomplete. The challenge for complete DCOP algorithms is to maximise efficiency. The challenge for incomplete algorithms is the assessment of approximate quality guarantees that allow to trade-off quality versus cost from an agent and/or a system designer perspective. Figure 1.1 shows the classification of quality guarantees into these three categories: optimality, agents' approximate guarantees, and system designer' approximate guarantees.

1.3 Contributions

In this dissertation we contribute to the challenges posed by quality assessment under resource-boundedness with frameworks, and algorithms, which enable the assessment of quality guarantees for the three categories shown in figure 1.1. The approach we follow is to exploit the structure of the problem to assess and bound high quality solutions.

First, as to *optimality guarantees*, we contribute with Action-GDL, a complete DCOP algorithm that generalises existing dynamic programming approaches by exploiting a more general representation of the problem. Second, as to *approximate guarantees*, we contribute with two frameworks for approximate quality assessment: *Divide-and-Coordinate*, which provides agent's quality guarantees, and *Region Optimality*, which computes system designer's quality guarantees. Additionally, from an algorithm design perspective, we formulate three novel incomplete DCOP algorithms: *DaCSA*, *EU-DaC* and $\mathcal{C}$*-DALO*. They all return suboptimal solutions with quality guarantees. Finally, we prove that region optimality is a valuable tool to bound the solutions of the Max-Sum algorithm, one of the most relevant algorithms in the DCOP literature. In what follows we describe the scope of these contributions in more detail.

Optimality guarantees

In the first part of this dissertation we focus on DCOPs for which we guarantee optimality. We present a novel complete DCOP algorithm, the so-called Action-GDL. Action-GDL improves the efficiency of some state-of-the-art complete DCOP algorithms by exploring a novel problem representation in terms of junction trees (Jensen and Jensen, 1994) which allows to better exploit, and with more flexibility, the structure of a DCOP. Although junction trees have been used before in other areas (e.g. find the most probable state in graphical models or decode error correcting codes), to the best of our knowledge, it is the first time that they are used for DCOP solving.

We define Action-GDL as an extension to the Generalized Distributive Law (GDL) algorithm (Aji and McEliece, 2000), a general message-passing algorithm that has been used by different communities under different names (e.g. Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms among others). This formal connections with the GDL framework allows us to show how Action-GDL unifies existing dynamic programming algorithms under GDL. Concretely, we prove that Action-GDL generalises two leading state-of-the-art dynamic programming DCOP algorithms, namely:

- the Dynamic Programming Optimization Protocol (DPOP) (Petcu and Faltings, 2005b), which uses a pseudotree representation of a DCOP; and
- the Distributed Cross-edged Pseudotree Optimization Procedure (DCPOP) (Atlas and Decker, 2007), which uses a cross-edge tree representation of a DCOP.

To prove such generalisation we formalise two different mappings between the space of problem representations used by these algorithms: (1) *a mapping from pseudotrees to junction trees*; and (2) *a mapping between cross-edge trees to junction trees*.

Finally, we provide theoretical and empirical results that characterise the improvement on efficiency of Action-GDL with respect to DPOP and DCPOP.

Approximate quality guarantees

In the rest of this dissertation we focus on domains in which achieving an optimal coordination is simply unaffordable.

Agent's quality guarantees

First, we define the DaC framework that defines an approach for approximate DCOP solving that allows agent's quality guarantees.

Intuitively, the DaC approach operates as follows. DaC agents *divide* the DCOP into simpler local subproblems that are individually solved by each agent. Then, because agents' solutions to their local problems may conflict, namely they can assign different values to very same variable, agents *coordinate* by exchanging local information and employ such information to update their subproblems. Thus, DaC agents iteratively *divide* and *coordinate* until finding an agreement, namely a set of subproblems whose local solutions assign the very same value to each variable of the problem.

When exploring the space of divisions, DaC agents exploit the structure of subproblems in order to: (i) assess good solutions; and (ii) bound the error of such solutions.

As to assessing good solutions, we formally prove that if all agents reach an agreement on a joint solution when optimizing their local subproblems, such solution is the optimal one. Therefore, agreement in DaC entails optimality, and hence agents can exploit local solutions, even in presence of conflicts, to generate solutions close to the optimal.

As to bound the error of these solutions, we prove that the sum of the values of local agent's solutions, that we shall refer to as the *value of a division*, bounds the quality of the optimal and hence, it allows agents to bound the error of their anytime solutions.

The DaC framework leads to the definition of different DaC algorithms, depending on the coordination information exchanged and the strategy used to reach an agreement. Concretely, in this dissertation we propose, and benchmark, two different DaC algorithms: a first one whose agents coordinate and update their subproblems based on their local solutions, the so-called *Divide and Coordinate Subgradient Algorithm* (DaCSA); and a second one whose agents coordinate and update their subproblems based on their utilities, that so-called *Egalitarian Utilities Divide-and-Coordinate* algorithm (EU-DaC). These results are significant because, in contrast with most DCOP incomplete algorithms that lack of quality guarantees, DaCSA and EU-DaC are scalable algorithms that can return anytime solutions with per-instance quality guarantees.

System designer's quality guarantees

Secondly, we contribute with region optimality, a framework that allows to assess quality guarantees that the system designer can use for a class of DCOP solutions, that we shall refer to as *region optimal solutions*. We define region optimal solution as a DCOP assignment, that is a set of agents' decisions, whose value can not be improved by changing the decision of any group of agents inside a region.

As part of region optimality we define mechanisms for computing system designer's quality guarantees on any region optimum. As argued in section 1.2, system designer's quality guarantees, besides being assessed at design time, need to be general enough to apply to any problem that agents may face at runtime. With this aim, we provide quality guarantees over region optimal solutions that apply to: (i) any problem instance; (ii) any problem with a particular graph structure; and (iii) any problem with a particular reward structure. The definition of region optimality allows us to explore a new dimension, namely the criteria used for defining regions. We show that one can benefit from exploring this larger space of local criteria by formulating a novel criterion to characterise regions, the so-called *size-bounded distance criterion*.

We present region optimality as an algorithmic-independent framework: quality guarantees are provided over region optimal solutions, independently of the algorithm employed to find them. Hence, we address the complementary algorithmic-design issue by defining $\mathcal{C}$-DALO, a generic region optimal DCOP algorithm that can find region optimal solutions for any specified arbitrary region. Since we can assess the cost of agents coordination in $\mathcal{C}$-DALO under different criteria, this generic region optimal algorithm, along with the corresponding quality guarantees, enables to trade-off quality versus coordination cost.

Finally, we claim that region optimality generalises and unifies the only two local optimality approaches in the literature that provide system designer's quality guarantees: k-size optimality (Pearce and Tambe, 2007) and t-distance optimality (Kiekintveld et al., 2010).

The last contribution of this thesis proves that region optimality framework is a valuable tool to bound the quality of Max-Sum solutions (Farinelli et al., 2008). Max-Sum is a state-of-the-art DCOP algorithm for which no quality guarantees are currently available. As a result, we are the first time to provide quality guarantees for Max-Sum at design time in general settings. Concretely, by means of region optimality we provide worst-case bounds on the quality of any Max-Sum solution (on convergence): (i) independently of the problem (problem-independent guarantees); and (ii) exploiting the graph structure of the problem (per-structure quality guarantees). These results shed light on the not well-understood behaviour of the Max-Sum algorithm and on the relationship between the quality of Max-Sum assignments and the structure of the problem. Moreover, it identifies new classes of graph structures for which we can provide significant guarantees on the quality of Max-Sum solutions.

This contribution is important for MAS coordination given the few DCOP algorithms that can provide system designer's quality guarantees. Moreover, since the Max-Sum algorithm, also known as loopy belief propagation (Pearl, 1988) or Max-Product (Aji and McEliece, 2000) algorithm, is one of the most used techniques for finding the

most probable state in graphical models, this contribution is also of interest to other areas such as statistical physics, computer vision or error-correcting coding theory, to name a few.

1.4 Guide to the Thesis

The remaining of this dissertation is organised as follows.

Chapter 2. We introduce the DCOP problem and a set of definitions and notations to define quality guarantees. Finally, we describe some motivating DCOP domains. The analysis of the potential of the Distributed Constraint Optimization framework to model problems that emerge in sensor networks has been published in:

- M.Vinyals, J. A. Rodriguez-Aguilar, J. Cerquides. *A Survey on Sensor Networks from a Multiagent Perspective*. The Computer Journal (2011) 54(3):455-470. 2010. first published online February 25, 2010. DOI:10.1093/comjnl/bxq018.

Chapter 3. We put in context our work with respect to the state of the art. In particular, we elaborate with the level of resources needed by these algorithms and the quality guarantees they provide. We also outline the limitations of current approaches when addressing quality assessment under resource-boundedness.

Chapter 4 introduces our contributions to designing complete DCOP algorithms related to the Action-GDL algorithm. The material contained in this chapter has been published in:

- M.Vinyals, J. A. Rodriguez-Aguilar, J. Cerquides. *Constructing a unifying theory of dynamic programming DCOP algorithms via the Generalized Distributive Law*. In Journal of Autonomous Agents and Multi Agent Systems (JAAMAS). Volume 22, Number 3, 439-464. DOI: 10.1007/s10458-010-9132-7.
- M.Vinyals, J. A. Rodriguez-Aguilar, J. Cerquides. *Action-GDL, a new complete algorithm for DCOPs*. In Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pages 1239-1240. Budapest, Hungary 5/2009.
- M.Vinyals, J.A Rodriguez-Aguilar, J. Cerquides. *Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs*. In Proceedings of the AAMAS'2009 Workshop on Optimization in Multi-Agent Systems (OPTMAS 2009). Budapest, Hungary 5/2009.

Chapter 5 includes our contributions related to the Divide-and-Coordinate framework. The material contained in this chapter has been published in:

- M.Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, J. Cerquides. *Divide and Coordinate: solving DCOPs by agreement*. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), pages 149-156. Toronto, Canada 5/2010.

- M.Vinyals, J. A. Rodriguez-Aguilar, J. Cerquides. *Divide-and-Coordinate by Egalitarian Utilities: turning DCOPs into egalitarian worlds*. In Proceedings of the AAMAS'2010 Workshop on Optimization in Multi-Agent Systems (OPTMAS 2010). Toronto, Canada 5/2010.

- M.Vinyals, J. A. Rodriguez-Aguilar, J. Cerquides. *Egalitarian Utilities Divide-and-Coordinate: Stop Arguing about decisions, let's share rewards*. In Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010), pages 1025-1026. Lisbon, Portugal 8/2010.

Chapter 6 includes our contributions related to the region optimality framework. The material contained in this chapter has been published in:

- M.Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, E. Bowring. *Quality guarantees for region optimal algorithms*. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011). Taipei, Taiwan, 5/2011. To appear.

- M.Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, E. Bowring. *Reward-based region optimal quality guarantees*. In Proceedings of the AAMAS'2011 Workshop on Optimization in Multi-Agent Systems (OPTMAS 2011). Taipei, Taiwan, 5/2011. To appear.

Chapter 7 includes the bounds on the solution quality for the Max-Sum algorithm.
The material contained in this chapter has been published in:

- M.Vinyals, J. Cerquides , A. Farinelli, J. A. Rodríguez-Aguilar. *Worst-case bounds on the quality of max-product fixed-points*. In Proceedings of the Neural Information Processing Systems (NIPS), pages 2325-2333, Vancouver, Canada, 12/2010. MIT press.

Chapter 8. We draw some conclusions and thoroughly describe paths to future research.

Finally, as a result of my master thesis *On the empirical evaluation of Mixed Multi-Unit Combinatorial Auctions* I also got the following publications:

- A. Giovannucci, J. A. Rodríguez-Aguilar, M. Vinyals, J. Cerquides. *Mixed Multi-unit Combinatorial Auctions for Supply Chain Management*. SIGecom Exchanges, Volume 7, Number 1, 2007.

- A. Giovannucci, J. Cerquides, U. Endriss, M. Vinyals, J.A. Rodríguez-Aguilar, B. Rosell. *A Mixed Multi-unit Combinatorial Auctions Test Suite*. In Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 09), pp. 1389-1390, 2009.

- M. Vinyals, A. Giovannucci, J. Cerquides , P. Meseguer, J.A. Rodríguez-Aguilar. *A Test Suite for the Evaluation of Mixed Multi-Unit Combinatorial Auctions*. In Journal of Algorithms, Volume 63, Issue 1-3, pp. 130-150, 2008. doi:10.1016/j.physletb.2003.10.071

- A. Giovannucci, M. Vinyals, J. A. Rodríguez-Aguilar, J. Cerquides. *Computationally-efficient Winner Determination for Mixed Multi-Unit Combinatorial Auctions*. In Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 08), pp. 1071-1078, 2008

- M. Vinyals, J. Cerquides. *On the Empirical Evaluation of Mixed Multi-Unit Combinatorial Auctions*. Agent-Mediated Electronic Commerce and Trading Agent Design and Analysis, Springer Berlin Heidelberg, Volume 13, p.135-150 (2008). doi:10.1007/978-3-540-88713-3

Chapter 2

Problem definition

This chapter introduces the Distributed Constraint Optimisation formalism (section 2.1), quality guarantees (section 2.2), and relevant domains for the application of DCOP techniques developed in this dissertation (section 2.3).

2.1 DCOP Definition

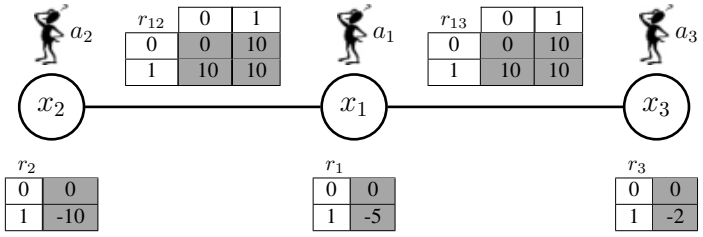
A Distributed Constraint Optimization Problem (DCOP) (Petcu, 2007; Modi et al., 2005) consists of a set of variables, each assigned to an agent, which must assign a discrete value to the variable. The set of values that an agent can assign to a variable correspond to individual actions that can be taken by this agent. Dependency relations exist between subsets of these variables that determine rewards to the agent team based on the combinations of values chosen by their respective agents. Solving a DCOP amounts to choosing values for the variables such that the solution quality is maximized.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. A relation on a set of variables $V \subseteq \mathcal{X}$ is expressed as a reward function $r_V : \mathcal{D}_V \rightarrow \mathbb{R}^+$, where $\mathcal{D}_V$ is the joint domain over the variables in V. Thus, a relation r_V assigns a utility value (reward) to each combination of values of its domain variables.
Formally, a DCOP is a tuple $\Phi = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where:

- $\mathcal{A}$ is a set of agents;
- $\mathcal{X}$ is a set of variables (each one assigned to a different agent);
- $\mathcal{D}$ is the joint domain space for all variables; and
- $\mathcal{R}$ is a set of utility/reward relations.

The objective function R is described as an aggregation over the set of relations. Formally:

$$R(d) = \sum_{r_V \in \mathcal{R}} r_V(d_V) \tag{2.1}$$



$$\mathbf{x}^* = \{\mathbf{x_1 = 1}, \mathbf{x_2 = 0}, \mathbf{x_3 = 0}\}, \mathbf{R}(\mathbf{x}^*) = \mathbf{15}$$

Figure 2.1: Example of a DCOP

where d is an element of the joint domain space $\mathcal{D}$ and $d_V \in \mathcal{D}_V$ contains the values assigned by d to the variables in V. The goal in a DCOP is to assess a configuration $x^* = \{x_i^* | x_i \in \mathcal{X}\}$ with utility $R(x^*)$ that maximizes the objective function in equation 2.1.

In a DCOP each agent receives knowledge about all relations that involve its variable(s). Although an agent can be in charge of one or more variables, hereafter, we assume that each agent a_i is assigned a single variable x_i. Therefore, across this thesis we will use the terms *agent* and *variable* interchangeably.

Binary DCOPs are usually represented by their constraint graphs, where nodes stand for variables and edges link variables that have some direct dependency (appear together in the domain of some relation). Figure 2.1 shows an example of a DCOP represented by its constraint graph. For instance, note that relation r_{12} is known by agent a_1, that controls variable x_1, and agent a_2, that controls variable x_2. In this context, the neighbours of some agent a are those that share some relation with a. Thus, in figure 2.1, a_2 and a_3 are neighbours of a_1 because a_1 shares relation r_{12} with a_2 and r_{13} with a_3. Additionally, each relation shows its reward in a table. Thus, agent x_2 has a reward of -10 to set its variable x_1 to 1 whereas a_1 and a_3 have a joint reward of 10 to set at least one of their variables to 1.

For the sake of simplicity, through in some parts of this dissertation we indulge a bit in notation as follows. Whenever there is no need to identify the domain, we simply use r to note a relation and $\mathcal{D}_r$ to denote the joint domain of its variables. This allows writing equation 2.1 as $R(d) = \sum_{r \in \mathcal{R}} r(d_r)$. where d is an element of the joint domain space $\mathcal{D}$ and d_r is an element of $\mathcal{D}_r$. Additionally, when focusing on binary DCOPs (those whose utility relations involve at most two variables), we will simplify notation and refer to unary constraints involving variable $x_i \in \mathcal{X}$ as r_i, and to binary constraints involving variables $x_i, x_j \in \mathcal{X}$ as r_{ij}.

2.2 Quality guarantees

Through this thesis we characterize algorithms depending on the quality guarantees they can provide over their solutions. Next, we provide formally definitions for these

quality guarantees.

A DCOP solution with quality guarantees is a solution x for which we can bound for its error with respect to the value of the optimal. We can classify these bounds on DCOP solutions values into two categories:

- *Absolute error bounds.* An absolute error bound $\delta \geq 0$ characterizes the error of the DCOP solution using an absolute distance with respect to the value of the optimal. Formally, a δ-absolute error bound over a solution x guarantees:

$$R(x) \geq R(x^*) - \delta \tag{2.2}$$

- *Relative error bounds*. A relative error bound $0 \leq \delta \leq 1$ characterizes the error of a DCOP solution as a fraction of the value of the optimal. Formally, a δ-relative error bound over a solution x guarantees:

$$R(x) \geq \delta \cdot R(x^*) \tag{2.3}$$

In general assessing a relative error bound is more desirable because they give a much intuitive idea of the magnitude of the error of a solution independently of the value of the optimal. The impact of absolute error bounds are highly influenced by the current value of the global optimum. For example, the same absolute error bound of 10 can lead to accurate solutions in a DCOP where the value of the optimal is 1000 but to poor solutions in a DCOP with an optimal solution value of 20. Relative error bounds capture this reality, by providing a percent error bound of 99% in the first case and of 50% in the second case.

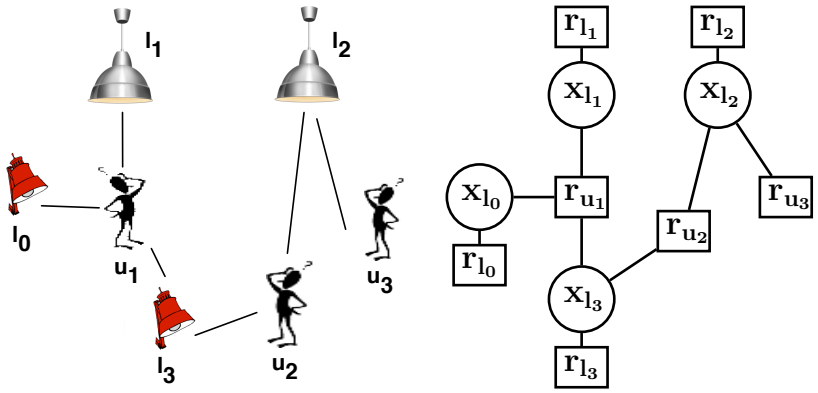
2.3 Motivating domains

There is a large class of multi-agent coordination problems that can be modeled in the DCOP framework. Examples include meeting scheduling, sensor networks, staff scheduling and power networks to name a few (Petcu and Faltings, 2008; Khanna et al., 2009; Vinyals et al., 2010; Maheswaran et al., 2004b). In the following, we will present in detail some examples that motivate the contributions of this thesis, namely indoor lighting control, gathering information in sensor networks, traffic light control and task allocation in disaster management scenarios.

2.3.1 Indoor lighting control

Increasing user comfort and reducing energy costs have always been two primary objectives of intelligent buildings (Finley et al., 1991). Recent works have proposed decentralised control approaches for this domain (Singhvi et al., 2005; Park et al., 2007).

Consider a building with n users $\{u_1, \ldots, u_n\}$ and m lamps $\{l_1, \ldots, l_m\}$. Users prefer different light levels that vary with sunlight conditions and when performing different tasks. Agents actuate over lamps to achieve specific desired light levels at different locations. The comfort of each user will depend on the state of nearby lamps, and nearby users may have conflicting desires. To achieve the setting of the lamps that are



(a) Lighting control problem. (b) DCOP model.

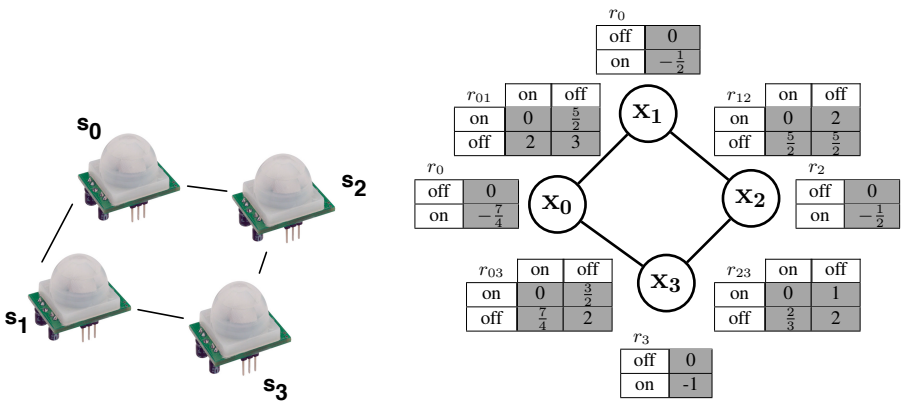
Figure 2.2: Lighting control problem modeled as DCOP. (a) shows a lighting control problem with three users and four lamps, and (b) DCOP model of the lighting control problem.

best for all of the users, we can specify for each user a reward function which encodes its local utility for a setting of the lamps. Higher utility function value means higher satisfaction for the user. DCOPs enacts the trade-off between these two objectives by allowing agents to coordinate lamp settings (i.e. intensity levels) to meet users comfort whereas minimising their consumption.

Definition 1. *The lighting control problem can be modeled as a DCOP* $\Phi = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ *such that:*

- $\mathcal{A} = \{a_1, \ldots, a_n\}$ *is a set of agents, one per each lamp;*
- $\mathcal{X} = \{x_1, \ldots, x_n\}$ *are decision variables that model the lamp settings;*
- $\mathcal{D}$ *is the joint settings for all lamps;*
- $R = \{r_{u_1}, \ldots, r_{u_m}, r_{l_1}, \ldots, r_{l_n}\}$ *is a set of utility functions, one per each user and one per each lamp. A user function* r_{u_i} *expresses the utility of user* u_i *for each possible configuration of the lamps. A lamp function* r_{l_i} *encodes the consumption costs of lamp* l_i*.*

Example 2.3.1.1 (Lighting control problem)**.** *See figure 2.2(a) for en example of lighting control problem with 4 lamps and 3 users. Figure 2.2(b) shows the DCOP model where round nodes stand for variables and square nodes stand for relations. The variables represent lamps and relations represent users preferences on the subsets of lamps variables. For example, user* u_1 *reports its preferences as a relation* r_{u_1} *that depends on three lamps variables* $x_{l_0}, x_{l_1}, x_{l_3}$*. Similarly, lamp* l_0 *encodes its consumption cost as a relation over the lamp variable* x_{l_0}*.*



(a) Sensor network problem. (b) DCOP model.

Figure 2.3: Information gathering modeled as DCOP. (a) shows an gathering information problem in a sensor network composed of four sensors, and (b) the DCOP model.

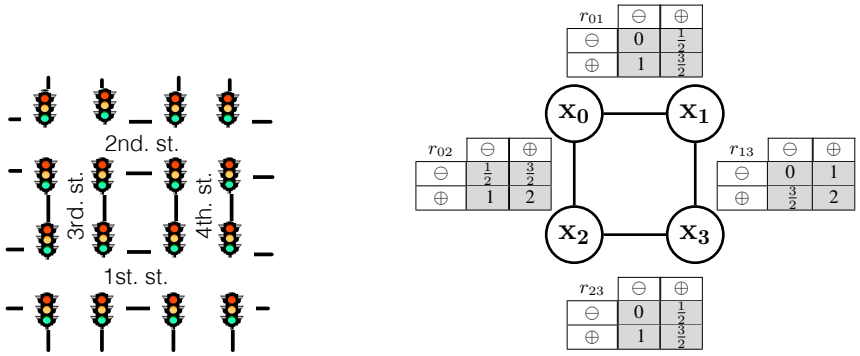
As argued in chapter 1, intelligent lighting control is an example of a domain where the characteristics of the environment and the architectural design principle of the building typically allows an efficient optimal coordination.

2.3.2 Gathering information in sensor networks

Gathering information in a fast, accurate and decentralised fashion is of vital importance in many domains, that include (but are not limited to) spatial phenomena (e.g. temperature or water contaminants) (Stranders et al., 2010; Krause et al., 2008), or targets moving within the environment (Vidal et al., 2001). In this context, the key challenge faced by sensors is the need to coordinate in order to maximise the amount of gathered information, since uncoordinated behaviour is likely to result in redundant sensor coverage of the environment and a waste of sensing resources.

This problem maps easily to a DCOP model. Consider a team of sensors $S = \{s_1, \ldots, s_n\}$. Each agent is assigned a sensor s_i for which it creates one variable x_i. A variable x_i models the possible actions (i.e. movements, measurement settings) of a sensor s_i. Then, the value of a set of sensors measurements $V \subseteq \mathcal{X}$ is mapped to a utility function r_V. The specification of this function depends on the application domain and the goals of the sensors within this application domain. For example, in (Stranders et al., 2009b) to predict spatial phenomena defines a value function stands for the entropy reduction that results from taking measurements for sensors at the positions specified for their variables.

Example 2.3.2.1 (Information gathering problem.)**.** *Please refer to figure 2.3(a) for en example of a sensor network for information gathering with four sensors. The sensors have fixed positions and have dependencies with the other sensors in a certain area around them (denoted with edges). For example, sensor* s_0 *has to coordinate with*



(a) Traffic light synchronization problem.

(b) DCOP model of the traffic light synchronization problem.

Figure 2.4: An traffic light synchronization problem. (a) The traffic light synchronization problem: there are four crossings with an square connection. (b) The DCOP model: each crossing has an associated variable that models the feasible plans for the crossing.

sensors s_1 and s_2 in order to avoid redundant measurements and minimise energy consumption. Figure 2.3(b) shows a DCOP model of the problem. The variables represent sensors and each variable's domain is composed of sensor's actions (to turn on or off). Unary relations stand for sensors costs, e.g. unary relation r_0 encodes a reward of $\frac{1}{2}$ for sensor x_0 to turn on. Binary relations stand for sensors dependencies, relation r_{01} encodes the rewards for the joint configurations of sensors s_0 and s_1.

As argued in chapter 1 for the specific case of environmental monitoring, this kind of sensor networks are typically composed of a large number of small-sized battery-operated sensors equipped with limited data processing and communication capabilities and hence, coordination to the optimal is in general not affordable under such severe conditions. Moreover, information gathering is a domain that allows to select among different on sensor positions or formations to trade-off quality versus cost.

2.3.3 Traffic light control

Consider the problem of synchronizing traffic lights in a city. Synchronization here means that adjacent traffic lights will be coordinated so that vehicles can traverse an arterial in one traffic direction, keeping a specific speed, without stopping. Each traffic light has a library of plans, each allowing the synchronization in different traffic direction. The specific objective of the scenario is to find the best signal plan configuration for the traffic lights.

This problem of traffic light synchronisation is formalized in (Junges and Bazzan, 2008) as a DCOP $\Phi = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ such that:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $A = \{a_1, \ldots, a_n\}$ are the sets of variables/agents, where n is the number of crossings;

- $\mathcal{D} = \{d_1, \ldots, d_n\}$ is the domain of the variables, representing the possible signals plans for each crossing agent;
- $\mathcal{R} = \{r_{1,1}, \ldots, r_{n,n}\}$ is the set of relations among the variables, each relation has a reward for a given pair of traffic plans of the two adjacent crossings. The reward that receive two agents to execute two plans in neighboring crossings is calculated in function of: (i) the degree in which these two plans synchronize; and (ii) the degree agents are coping with the volume of vehicles in that direction (fraction of vehicles in that lane). Notice that whereas (i) is fixed (ii) vary with the traffic conditions.

Example 2.3.3.1 (Traffic light control.)**.** *See figure 2.4(a) for an example of traffic light control problem with four crossing with an square connection. Figure 2.4(b) shows the DCOP model for this problem. The variables represent crossing, and each one has as domain the possible plans for the traffic lights in this crossing. Relations link crossing variables that need to be synchronised.* $\oplus$ *means that plan is synchronized and agrees with the direction of higher traffic volume.* $\ominus$ *means that plan is synchronized in a direction other than that of higher traffic volume. For example, by means of relation* r_{13} *when the first and third crossing are synchronized in the direction of higher traffic volume receive a reward of 2, whereas when only the third crossing is synchronised in that direction the reward is 1.*

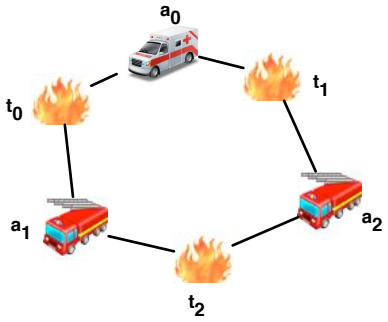
As discussed in chapter 1, traffic control is an example of a domain where rewards vary continuously but where the structure of dependencies among (crossing) agents known and fixed and hence, can be exploited on coordination to trade-off quality versus cost.

2.3.4 Task allocation in distributed scenarios

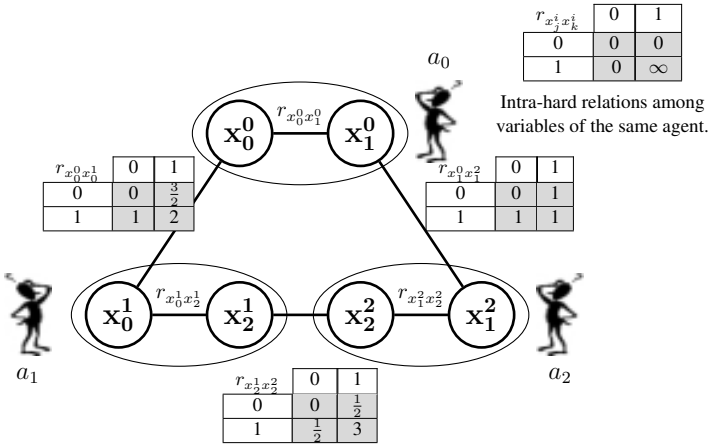
In a major disaster, such as earthquakes or terrorist attacks, rescue agents are required to perform a number of rescue tasks (i.e. rescuing civilians or extinguishing fires) in different parts of the affected area. Each task requires a given level of effort and may have to be performed by a certain deadline (otherwise, humans can die or some infrastructures be devastated). In order to complete as many tasks as possible by their deadline, agents need to form teams or coalitions of multiple responders. This is because no single agent will have all the resources needed to perform a task (i.e. save all the victims or extinguish the fires). Hence, it is critical that these processes of coalition formation and management are effectively enacted through coordination.

This coordinating of emergency responders in disaster management can be conveniently framed as a DCOP (Macarthur et al., 2010; Ramchurn et al., 2010). Indeed, there are many ways in which we can formalise this task allocation problem as a DCOP, depending on what we choose to represent with variables and how we define the relations among them. We detail in the following one possible way to cast this problem to a DCOP model.

Consider a disaster management scenario with a the set of rescue agents $a_1, \ldots, a_n$ and a set of rescue tasks to be performed denoted as $t_1, \ldots, t_m$. Each rescue agent a_i creates for each task $t_j \in T_i$ that it can execute a local binary variable x_j^i that models if



(a) Traffic light synchronization problem.



(b) DCOP model of the traffic light synchronization problem.

Figure 2.5: An task allocation problem. (a) The task allocation problem: there are three rescue agents and three fires. (b) The DCOP model.

a_i is assigned ($x_j^i = 1$) or not ($x_j^i = 0$) to task t_j. Then a_i connects all variables x_j^i from its local problem with a relation r_{a_i} that ensures that a_i is only allocated to one task, assigning an infinite cost to non-feasible combinations. Finally, each rescue task t_j creates a relation r_{t_j} that connects all variables x_j^i encoding its degree of accomplishment for each combination of agents assigned to it.

Example 2.3.4.1 (Task allocation problem)**.** *We give an example environment in figure 2.5(a), which contains three rescue agents, and three tasks which they must complete (in this case, fires to be extinguished). Figure 2.5(b) shows a DCOP model of the problem. Each rescue agent creates one variable for rescue task it can participate to. For example, rescue agent* a_0 *creates two variables* x_0^0,x_1^0 *corresponding to the two task fires* t_0, t_1 *it can contribute to extinguish. Variable* x_0^0 *it is linked to variable* x_0^1 *through relation* $r_{x_0^0 x_0^1}$*. Thus, relation* $r_{x_0^0 x_0^1}$ *encodes the level of accomplishment of task* t_0 *depending if* a_0 *and/or* a_1 *are assigned to it. Observe that in figure 2.5(b) the reward*

table of $r_{x_0^0 x_0^1}$ that if a_0 and a_1 are assigned to task t_0 their receive a joint reward of 2.

As discussed in chapter 1, disaster management is clearly one of the domains where agents cannot afford the risk that their coordination degrades below a certain level due to the human impact and structural factor it can have. Additionally, optimal coordination is too time consuming in this continuously changing domain with strict time deadlines and communications restrictions (communications can be overloaded or affected by the same disaster).

Chapter 3

Related work

This chapter gives an overview of the state-of-the-art in DCOP algorithms and places the contributions of this thesis with the existing related work.

Focusing on the problem of providing efficient algorithms that can provide quality guarantees, next, in sections 3.1 and 3.2 we review existing DCOP algorithms classifying them on complete and incomplete algorithms respectively. We further classify DCOP algorithms based on the approach they follow to solve a DCOP. As to complete algorithms, we classify them on *fully decentralised approaches*, search and dynamic programming, and *partially centralised approaches*. As to incomplete DCOP algorithms, we classify them on *decision-based*, in which agents coordinate based on exchanging their local decisions, and *GDL-based*, in which agents coordinate exchanging utilities following GDL update rules. Afterwards, in section 3.3 we outline some active lines of work in the DCOP literature that are complementary but beyond the scope of this thesis. Finally in section 3.4 we summarize the limitations of the state-of-the-art DCOP algorithms to overcome the challenges listed in chapter 1.

3.1 Complete DCOP algorithms

As mentioned in chapter 1, complete DCOP algorithms are guaranteed to find an optimal solution for a DCOP, and are suitable for domains in which agents can afford the complexity of optimality. Given that, the challenge of designing complete DCOP algorithms consist in their efficiency, namely the amount of communication, computation, and parallelism required by agents at run-time.

Next, in section 3.1.1 we review complete fully decentralised DCOP algorithms, namely algorithms where each agent operates totally decentralised dealing only with local information. Then, in section 3.1.2, we review an alternative approach to maximising efficiency based on the partial centralisation of the DCOP problem in some single agent.

	Approximate Guarantees				
	Time		Specificity		
	Design	**Run**	**Problem-independent**	**Per-Class**	**Per-Instance**
ADOPT[1]	✓		✓		
BnB-ADOPT[1]	✓		✓		
A-DPOP	✓		✓		
DSA					
MGM-1					
Max-Sum					
Bounded Max-Sum		✓			✓
MGM/SCA-{2,3}	✓		✓	✓	
k- DALO	✓		✓	✓	
t- DALO	✓		✓	✓	

Agent's quality guarantees
System designer's quality guarantees

[1] ADOPT and BnB-ADOPT here refer to the bounded-error approximation extensions of the respective complete algorithms.

Table 3.1: Quality assessment landscape for incomplete DCOP algorithms. Guarantees are characterised based on two dimensions: time at which they are available and specificity.

3.1.1 Fully decentralised approaches

State-of-the-art complete algorithms to solve DCOPs adopt two main techniques: dynamic programming and search. Even though both approaches have an exponentially increasing coordination overhead, due to optimality guarantees, they differ on the nature of such overhead. Search algorithms require linear-size messages, but an exponential number of messages. Dynamic programming algorithms only require a linear number of messages, but their complexity lies on the message size, which may be very large. Table 3.2 depicts DCOP algorithms landscape where algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes). Observe that search and dynamic programming are included as approaches in the upper horizontal axis for complete algorithms.

Search algorithms

DCOP search algorithms use search strategies to search for an optimal solution. The advantage of search algorithms is that they require polynomial memory. Their downside is that they may produce a very large number of small messages, resulting in large communication overheads. Although several search DCOP algorithms have been proposed in the literature (SSB (Hirayama and Yokoo, 1997), NCBB (Chechetka and Sycara, 2006), AFB (Gershman et al., 2009), ADOPT (Modi et al., 2005)), possibly ADOPT along with its extensions are the most representative ones (Maheswaran et al., 2004b; Ali et al., 2005; Silaghi and Yokoo, 2006; Gutierrez and Meseguer, 2010). In what follows we review the key ideas behind ADOPT operation, and the problem representation it uses to solve DCOP problems.

ADOPT's popularity stems from being the first decentralized search DCOP algorithm that allows to operate asynchronously. ADOPT compiles a DCOP into a pseudotree structure (Freuder and Quinn, 1985), which it uses as a hierarchy to communicate value and cost messages. In the worst-case, the number of messages exchanged in ADOPT is exponential to the depth of the selected pseudotree. Since finding the pseudotree with minimal depth is NP-Hard, in practice, ADOPT-based approaches rely on greedy heuristics (Modi et al., 2005; Chechetka and Sycara, 2005) to assess a low-depth pseudotree.

Each agent in ADOPT maintains lower and upper bounds for the subtree rooted at its node. These are computed by exchanging messages with agent's neighbours in the pseudotree. ADOPT agents exchange three kinds of messages: cost messages, which are propagated up the tree, and threshold and value messages, which are propagated down. Value message contain the variable assignment selected by an agent using best-first search by assigning the value with smaller lower bound. Each ADOPT agent operates asynchronously by changing its variable's values whenever it detects that has smaller lower bounds, although they are not guaranteed to be better. Abandoning partial solutions before proving their suboptimality makes agents revisit several times some of the previously explored partial solutions. As a solution, a backtracking threshold is used for reducing the need for backtracking while maintaining a low memory profile.

Another important characteristic of ADOPT, not shared with other search algo-

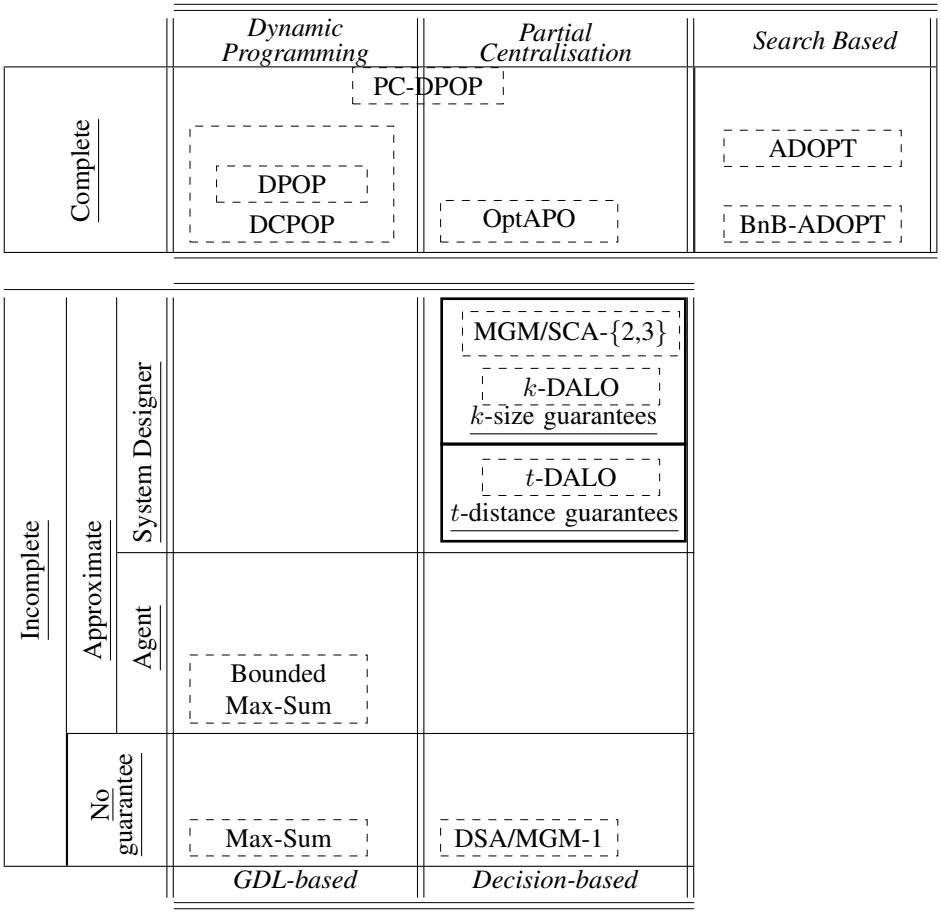


Table 3.2: DCOP algorithms landscape. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

rithms, is the option of running as a bounded complete algorithm, allowing users to specify an absolute error bound, prior to the algorithm's execution. The advantage of running ADOPT as a bounded incomplete algorithm is that it is more likely to find solutions for which it provides problem-independent guarantees faster, providing a way to trade-off solution quality versus cost. Table 3.1 characterise the quality guarantees provided by incomplete DCOP algorithms, including the bounded ADOPT version that provides problem-independent guarantees at design time. However, observe that these guarantees do not satisfy the requirements listed in chapter 1 neither for agents not for the system designer. On the one hand, agents can not use these guarantees to bound the quality of their solution at runtime because they apply on the solution reached on convergence and there is no guarantee or bound on the computation time or commu-

nication overhead required to achieve such convergence. On the other hand, since this bound does not allow to exploit any characteristic of the problem, the trade-off that can execute the system designer is limited and very likely to be inaccurate.

A recent extension of ADOPT is BnB-ADOPT (Yeoh et al., 2010), which is based on the same message passing and communication framework of ADOPT, but changes the best-first search strategy for a depth-first branch-and-bound search. Although having the same worst-case complexity, exponential to the number of agents, BnB-ADOPT has been shown to provide optimal solutions up to one order of magnitude faster than ADOPT. Another advantage of BnB-ADOPT with respect to ADOPT is that it allows to specify a user relative error bound, instead of an absolute bound. As mentioned in section 2.2, usually relative error bound give a more intuitive idea of the magnitude of the error of a solution. As summarised in table 3.1, BnB-ADOPT provides problem-independent guarantees at design time, and likewise ADOPT bounded version, does not satisfy the requirements listed neither for agents nor for the system designer.

Table 3.2 includes ADOPT and BnB-ADOPT in the DCOP landscape as search based complete DCOP algorithms.

Dynamic Programming algorithms

As to dynamic programming approaches, the Dynamic Programming Optimization Protocol (DPOP) (Petcu and Faltings, 2005b) was the first algorithm based on dynamic programming to perform variable elimination on a pseudotree in a distributed fashion. Hence, as ADOPT-like search algorithms, DPOP agents distributedly compile the DCOP into a pseudotree structure.

DPOP operates in two main phases. During the first phase, each agent propagates messages that contain information about its rewards up to the tree by joining messages from children and its local rewards. During the second phase, each agent chooses an assignment for its variable by joining the assignments from its parent and propagates them down to the tree.

The number of messages sent between agents is only linear in the number of agents and the complexity of DPOP depends on the size of the largest computation and message exchanged during the second phase. This complexity directly corresponds to the induced width of the pseudotree (Petcu and Faltings, 2005b). Since finding the minimum induced width pseudotree is NP-Hard, DCOP efficiency relies on using distributed heuristics to generate low-width pseudotrees (Petcu, 2007).

Many extensions have been defined to DPOP with the aim of providing different trade-offs (e.g. MB-DPOP (message-size vs memory), M-DPOP (extensions to consider self-interested agents)). One of this extensions, A-DPOP (Petcu and Faltings, 2005a), consists in a parametrized approximation that provides a trade-off between solution quality and computation complexity, similarly to ADOPT bounded complete approximations. A-DPOP allows user to specify a user relative error bound on the solution value, and tries to reduce the computation and communication overhead as much as possible to find a solution that is guaranteed to be within such bound. Table 3.1 includes A-DPOP as incomplete DCOP algorithm that provides problem-independent guarantees at design time. However, likewise ADOPT and BnB-ADOPT bounded versions,

these guarantees do not satisfy the requirements neither for agent not for the system designer.

Because the complexity of all these DPOP-like algorithms relies on finding a good problem representation, in (Atlas and Decker, 2007) Atlas and Decker provide a generalization of DPOP, the Distributed Cross-edged Pseudotree Optimization Procedure (DCPOP) by using a cross-edge tree representation of the problem. Their experimental results show that using a cross-edge tree representation leads to important benefits in terms of efficiency with respect to pseudotrees. Table 3.2 shows DPOP and DCPOP (which subsumes DPOP) in the DCOP landscape as complete dynamic programming algorithms.

3.1.2 Partial centralized approaches

Instead of providing a new technique to solve DCOPs, partial centralisation consists in exploring the gap between centralized and distributed techniques and their effects on efficiency. Some argue that fully decentralized algorithms often require excessive amounts of communication when applied to complex problems (Davin and Modi, 2005; Petcu et al., 2007; Mailler and Lesser, 2006). OptAPO (Optimal Asynchronous Partial Overlay)(Mailler and Lesser, 2006) is the first algorithm that uses a dynamic, partial centralization.

OptAPO agents dynamically discover difficult subproblems, namely parts of the global problem with strong dependencies across agents. It centralises these subproblems by introducing the role of a mediator. When an agent acts as a mediator, it computes a solution for a part of the overall subproblem, using any current state-of-the-art centralised solver, and recommends value changes to agents that depend on the mediator. An OptAPO agent retains its autonomy by either having the possibility of refusing solutions posed by a mediator or taking over as mediator itself if it does not agree with a proposed solution.

The advantage, and novelty, of OptAPO consists in allowing agents to explore the level of centralisation more appropriate to the problem in a dynamic way. The downside of this approach is that depending on the difficulty of the problem, it can lead to several mediators centralizing most of the problem and carrying out most of the computation in a redundant way. In these cases OptAPO shows a poor scalability with respect to fully decentralised approaches (Davin and Modi, 2005; Petcu et al., 2007). Furthermore, the asynchronous and dynamic nature of the centralisation process makes impossible to theoretically characterize the complexity of this approach depending on the problem structure.

Finally, this idea of partial centralisation has also been explored by Petcu and Faltings in (Petcu et al., 2007) by formulating PC-DPOP, a hybrid algorithm that extends DPOP to trade-off the number of messages versus partial centralisation. Therefore, table 3.2 includes OptAPO as a partially centralised complete algorithm and PC-DPOP as an hybrid complete algorithm between partially centralised and dynamic programming approaches.

3.2 Incomplete DCOP algorithms

As discussed in chapter 1, there are resource-bounded domains for which complete DCOP algorithms are simply unaffordable. Incomplete DCOP algorithms represent a good alternative in these domains because they sacrifice optimality to obtain fast anytime solutions. Typically, incomplete DCOP algorithms have involved loosing some quality guarantees.

We classify incomplete DCOP algorithms in two groups based on their operation: decision-based and GDL-based. The first category stand for algorithms in which individual or a group of agents coordinate their decisions based on the decisions of their neighbours. The second category corresponds to algorithms based on the Generalized Distributive Law (Aji and McEliece, 2000), a general framework that has been successfully applied to many areas in computer science and information theory. In contrast with decision-based, in GDL-algorithms agents coordinate by exchanging the utilities to take some decision instead of decisions themselves.

3.2.1 Decision-based algorithms

The key idea behind decision-based algorithms is that a group of agents optimize their joint decision given the decisions of their neighbours.

As we can observe in table 3.2 the Distributed Stochastic Algorithm (DSA) (Fitzpatrick and Meertens, 2002) and the Maximum Gain Message (MGM) (Maheswaran et al., 2004a) algorithms fall in this category as decision-based algorithms that do not provide any guarantee on their solutions. Both MGM and DSA have very similar operation in which each agent optimize their decision individually, in groups of a single agent. As to MGM, each agent acts deterministically by choosing at each iteration the value for its variable that maximizes its reward given the values chosen by its neighbours at the last iteration. DSA differs from MGM on introducing a stochastic element in the agent decision process. Thus, an agent decides randomly whether to change its current decision for one with better reward. In general, stochastic approaches reduce the number of messages exchanged but not always improve the quality of the solution reached on convergence (Pearce et al., 2008).

MGM and DSA algorithms are limited by their ability to aggregate information about their neighbours because they restrict to local (agent) optimisations. Therefore, a natural extension consists in considering algorithms that coordinate more than one agent instead of focusing on individuals. That is the approximation explored by the k-size (Maheswaran et al., 2004a) algorithms, which operates by optimising groups of k agents, and t-distance (Kiekintveld et al., 2010) algorithms, which operates by optimising groups of agents in a distance t from a central agent. The first k-size optimal algorithms in the literature were: (i) MGM-2 and MGM-3, extensions of the MGM-1 algorithm to consider groups of 2 and 3 agents; and (ii) SCA-2 and SCA-3 algorithms, 2- and 3-size optimal stochastic algorithms based on DSA (Pearce et al., 2008).

Later, in (Kiekintveld et al., 2010) Kiekintveld et al. proposed DALO an asynchronous local search algorithm that allows either k-size (k-DALO) or t-distance solutions (t-DALO) for arbitrary values of t and k. Their empirical results showed that: (i) k-DALO outperforms MGM and DSA when looking for size optimal solutions of the

same size; and (ii) exploring t-distance as alternative criterion to characterise groups leads to significant empirical benefits in some problem settings.

A distinctive property of k-size and t-distance optimal algorithms is that they are the only incomplete DCOP algorithms that can provide quality guarantees such that satisfy the requirements for the system designer, namely they are problem-independent and per-class guarantees assessed at design time. First, the works of Pearce and Tambe (Pearce and Tambe, 2007) and later of Kienkintveld et al.(Kiekintveld et al., 2010) defined problem-independent quality guarantees for solutions of k-size and t-distance optimal algorithms respectively. Second, in the same works, authors extended these guarantees to consider the structure of agent's dependencies (per-structure quality guarantees). As argued in chapter 1, these per-structure quality guarantees not only help to determine the appropriate algorithm considering the problem structure, but also to select an appropriate graph structure to model the problem. Finally, the work of Bowring et al. (Bowring et al., 2008) extended k-size optimal guarantees to exploit the characteristics of the rewards (per-reward quality guarantees). Concretely, they show how to compute tighter guarantees by assuming a ratio between the least minimum reward to the maximum reward among relations. To the best of our knowledge, these per-reward quality guarantees have only been defined over the solutions returned by k-size algorithms, and not extended to those of t-distance algorithms. In summary, table 3.1 classifies MGM/SCA-$\{2,3\}$, k-DALO, and t-DALO, as algorithms that return system designer's quality guarantees (green rows) namely problem-independent and per-class guarantees assessed at design time. Table 3.2 also includes these algorithms in the DCOP landscape as decision-based incomplete DCOP algorithms that provide system designer's guarantees, concretely k-size guarantees and t-distance guarantees.

3.2.2 GDL-based algorithms

The Generalized Distributive Law (GDL) (Aji and McEliece, 2000) is a general message passing algorithm synthesis of the work in many fields such as information theory, signal processing or statistics. It includes as special cases Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms among others. In GDL approaches, agents operate by exchanging the information related to set their variables to particular states.

The first GDL-based algorithm formulated in the DCOP community was the Max-Sum algorithm (Farinelli et al., 2008), an application of the well-known Loopy Belief Propagation (Pearl, 1988) or Max-Product algorithms (Aji and McEliece, 2000), though applied to the problem of maximizing the social welfare in Multi-Agent coordination. Therefore, in contrast with decision-based algorithms, in Max-Sum agents operate by exchanging the utilities to take some decision, instead of the decision themselves.

Max-Sum stands for the iteratively, approximate version of GDL. It runs over the DCOP constraint graph instead of compiling the problem into another representation over which GDL can guarantee optimality (at expenses of increasing their cost). As a result, Max-Sum is an incomplete algorithm that is guaranteed to converge to the optimum in acyclic constraint graphs, but with no guarantees of convergence and solution quality on general graphs. However, the solution that Max-Sum returns on convergence has been characterized to be neighborhood maximum, rather than a simple local maxi-

mum (Weiss, 2000), which makes expect solutions of more quality than those of local optimal algorithms.

On the one hand, the popularity of Max-Sum stems from its good empirical performance. Several works have reported good empirical results on DCOP datasets by benchmarking it against other state-of-the-art DCOP algorithms (Kok and Vlassis, 2006; Farinelli et al., 2008; Rogers et al., 2011). These results are not surprising given the empirical results reported for the iterative approximate version of GDL in other areas (Murphy et al., 1999; Frey et al., 2001b,a). On the other hand, its downside lies in that its behaviour is not well understood due to the lack of quality guarantees, only defined in very restricted cases.

To overcome this lack of quality guarantees, Farinelli et al. propose in (Farinelli et al., 2009) a bounded version of Max-Sum based on exploiting its optimality on acyclic constraint graphs. Following this idea, given a DCOP, bounded Max-Sum runs over an induced tree of its constraint graph. This induced tree is distributedly built by agents by ignoring dependencies between the relations and variables that have less impact on solution quality guarantee. Then, agents run Max-Sum on this induced tree obtaining the optimal solution for this approximate problem representation. By considering the particular rewards of the ignored edges and running Max-Sum over the induced tree, bounded Max-Sum allows quality guarantees by bounding the error of its solution. As summarised in table 3.1, these quality guarantees are per-instance quality guarantees that agents can use over their solution at runtime (unlike bounded ADOPT approaches and A-DPOP bounded Max-Sum is guarantee to converge in a linear number of operations). Hence, bounded Max-Sum quality guarantees satisfy the requirements of agents to trade-off quality versus cost at runtime.

Table 3.2 includes both Max-Sum and bounded Max-Sum algorithms as GDL-approaches in the DCOP landscape. As to quality guarantees, Max-Sum falls in the category of algorithms that can not provide any guarantee over their solutions whereas bounded Max-Sum returns agent's quality guarantees.

3.3 Beyond the scope of this thesis

Distributed Constraint Optimisation is a very active area of research that has led to multiple parallel lines of work. On the one hand, many works have focused on extending the original DCOP formulation to consider uncertainty (Taylor et al., 2010; Tambe et al., 2005; Laut and Faltings, 2009), continuous variables (Voice et al., 2010; Stranders et al., 2009a), multi-objective optimization (Bowring et al., 2006), privacy (Greenstadt, 2009), self-interested agents (Petcu et al., 2008), or adaptation on dynamic environments (Sultanik et al., 2009; Khanna et al., 2009). On the other hand, further research has focused on optimizing existing algorithms for particular problems domains, such as Fast Max-Sum (Macarthur et al., 2010), which optimizes Max-Sum algorithm for distributed task allocation or LA-DPOP (Scerri et al., 2005), which uses a token-based approach similar to DSA to minimize communication in distributed task allocation with high overlapping team functionality.

In contrast, the contributions presented in this thesis are formulated over the standard DCOP formulation, as introduced in chapter 1, without tailoring to any specific

domain. Hence, although the above-mentioned works are beyond the scope of this thesis, our contributions to the design of efficient DCOP algorithms with quality guarantees are likely to provide them with useful insights for further extensions.

3.4 Limitations of the current approaches

In this section we analyse which are the limitations and drawbacks of the current approaches on providing efficient DCOP algorithms with quality guarantees.

3.4.1 On exploring efficient problem representations for optimal DCOP solving

The efficiency of DPOP-like dynamic programming and ADOPT-like search approaches highly depends on finding a good problem representation. As reviewed in section 3.1, the space of representations explored by both approaches is the space of pseudotrees. The work of Atlas and Decker on extending DPOP algorithm to handle cross-edge trees, a generalisation of pseudotrees, opens a new avenue of research, namely how to improve the efficiency of complete algorithms by exploiting more general problem representations. However, there are some fundamental questions that remain open and that must explored to progress this line of work, namely:

- Is there any further problem representation that can be exploit by complete approaches?

- Can we theoretically or/and empirically characterize the potential improvement to explore such problem representation?

In the chapter 4 of this dissertation we contribute to answer these questions for the particular case of the dynamic programming DCOP algorithms. Concretely, we formulate Action-GDL a novel complete DCOP algorithm that generalises DPOP and DCPOP, unifying them under the GDL framework, by handling a more general problem representation, based on a junction tree.

3.4.2 On assessing agents' quality guarantees

Notice that in table 3.1 the only incomplete DCOP algorithm that can provide quality guarantees such that satisfy the requirements of agents, as listed in chapter 1, is the bounded Max-Sum algorithm. Hence, we identify a clear need for developing new DCOP approximations that can provide agent's quality guarantees, that as argued in chapter 1 reduce the uncertainty from an agent perspective.

This thesis addresses this issue on chapter 5, by proposing a new family of incomplete DCOP algorithm that provide agent's quality guarantees based on a novel approach, the so-called *Divide-and-Coordinate* approach.

3.4.3 On characterising local optimal solutions that allow system designer's quality guarantees

As shown in table 3.1, the only incomplete DCOP algorithms that provide guarantees such that satisfy system designer's requirements are k-size optimal (k-DALO, MGM-$\{2,3\}$) and t-distance optimal (t-DALO) algorithms. These quality guarantees are those defined by k-size and t-distance optimality over k-size and t-distance local optimal solutions respectively.

k-size and t-distance optimality establish an important research direction by defining algorithmic-independent approximate guarantees for two classes of DCOP local optima, namely those characterised by size or distance criteria. However, the inability of current approaches to assess quality guarantees for a larger set of optima defined over any arbitrary criteria has limited so far their impact and applicability of these approaches.

This limitation poses some fundamental questions that must explored to progress this line of work, namely:

- Can we define analogous approximate quality guarantees for a larger set of local optima, namely optima defined over any arbitrary criteria?
- Does a better criteria (beyond size and distance) exist that offer better guarantees, faster algorithms or more fine-grained control of the trade-off?

This dissertation addresses both questions in chapter 6 by defining region optimality, a general framework that defines system designer's quality guarantees for solutions using any arbitrary criteria, beyond k-size and t-distance optimality.

3.4.4 Quality guarantees for the Max-Sum algorithm

When looking at table 3.1, we observe that DSA, MGM-1 and Max-Sum are algorithms that lack of any quality guarantee over their solutions. DSA and MGM-1 has been characterised in the k-size optimality framework as 1-size algorithms. Thus, since the optimality of their solutions is restricted to individual agents, it is not surprising that they lack of quality guarantees. However, that is not the case of the Max-Sum algorithm. As argued in section 3.2, solutions achieved by Max-Sum on convergence are known to be a neighbourhood maximum rather than a simple local maximum. These theoretical results are consistent with the good empirical performance of Max-Sum. Therefore, this lack of quality guarantees hinters the characterisation of the kind of problems for which we can expect such good performance from those in which Max-Sum converge to much more worse solutions. Hence, to overcome this limitation we need to address the following question:

- Can we take advantage of the characterisation of Max-Sum solutions on convergence as neighborhood maximum to provide guarantees on their quality?

In chapter 7, we address this question by defining system designer's quality guarantees for Max-Sum solutions.

Chapter 4

Action-GDL: Extending GDL to solve DCOPs

In this chapter we face the problem of designing efficient, complete DCOP algorithms for domains with enough resources to search for optimal coordination. As analysed in section 1.2, the challenge in these domains is how to design efficient, complete DCOP algorithms, namely algorithms that minimize the communication/computation required by agents at runtime.

To achieve that purpose we propose a novel, complete DCOP algorithm, the so-called Action-GDL. Action-GDL is an extension to the Generalized Distributive Law (GDL) algorithm (Aji and McEliece, 2000), a general message-passing algorithm that has been inadvertently used by different communities under different names (e.g. Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms). In particular, in this chapter we observe that this is also the case for the Cluster Tree Elimination algorithm in the constraint community (Dechter, 2003).

Then, we show how Action-GDL advances the state-of-the-art of complete DCOP algorithms from an analytical and from an empirical perspective.

From an analytical perspective, we relate Action-GDL with existing complete DCOP algorithms in the literature by showing that it generalizes DPOP (Petcu and Faltings, 2005b) and DCPOP (Atlas and Decker, 2007). By doing so, we provide a unifying theory for dynamic programming DCOP algorithms based on GDL. These algorithms include optimal DCOP algorithms, such as DPOP and DCPOP, as well as incomplete DCOP algorithms such as Max-Sum (Farinelli et al., 2008). Moreover, this unifying perspective provided by Action-GDL builds a bridge with a wealth of theoretical results for GDL from which the DCOP community may benefit.

From a pragmatical perspective, we show that Action-GDL allows to solve DCOPs more efficiently by exploiting an extended set of problem representations based on junction trees. First, we characterise when Action-GDL can outperform DPOP by moving from pseudotrees, used by DPOP, to junction trees. Second, we show empirical evidence of the benefits in terms of efficiency that Action-GDL can provide with respect

to DCPOP, and by extension of DPOP.

This chapter is organized as follows. Section 4.1 introduces some notation used throughout the rest of the chapter. Section 4.2 describes some background on GDL, junction trees, and their relation with the Cluster Tree Elimination algorithm. Section 4.3 introduces Action-GDL, assesses its complexity and defines a distributed algorithm to map a DCOP into a distributed junction tree. Section 4.4 proves that DPOP and DCPOP are particular cases of Action-GDL. Section 4.5 theoretically characterises when Action-GDL can outperform DPOP and formulates a distributed post-processing heuristic to optimize junction trees. Section 4.6 compares the efficiency of Action-GDL with DPOP and DCPOP. Finally, section 4.7 draws some concluding remarks.

4.1 Notation

Next, we define the functions and operators that we shall employ henceforth through the rest of the chapter. For the sake of simplicity we assume binary utility relations involving two variables, although all the results in the chapter are valid for any arity.

Definition 2 (Scope)**.** *The scope function returns the domain variables of a given set of relations. Ex:* $Scope(\{r_{12}\}) = \{x_1, x_2\}$, $Scope(\{r_{12}, r_{24}\}) = \{x_1, x_2, x_4\}$.

Definition 3 (Complementary variables)**.** *Given a set of variables* $V \in \mathcal{X}$ *and a relation* r*, we define the complementary variables of* V *by* r *as the set of variables in* r *that are not in* V*. Formally,* $\bar{V}^r = Scope(r) \setminus V$.

Definition 4 (Utility message)**.** *A message from agent* x_i *to agent* x_j *is a utility message* $\mu_{ij}(V)$ *over* $V \subseteq \mathcal{X}$*, if the information sent is a relation over* V.

Definition 5 (Assignment)**.** *Given a set of variables* $V \in \mathcal{X}$*, an assignment* σ *over* V *sets a value to each variable* $x_k \in V$ *and sets free the remaining variables. Given* $V' \subset V$*, we note by* $\sigma_{V'}$ *the projection of* σ *to* V'*, that is, the assignment that sets the same value as* σ *for the variables in* V'*. Ex:* $V = \{x_1, x_3\}$*,* σ *an assignment over* V *with* $\sigma(x_1) = 1$*,* $\sigma(x_3) = 0$*.* x_2 *and* x_4 *are free in* σ*. If* $V' = \{x_1\}$ *we have that* $\sigma_{V'}(x_1) = 1$*.* x_2*,* x_3 *and* x_4 *are free in* $\sigma_{V'}$.

Definition 6 (Value message)**.** *A message from agent* x_i *to agent* x_j *is a value message* $\sigma_{ij}(V)$ *over* $V \subseteq \mathcal{X}$ *if the information sent is an assignment over* V.

Definition 7 (Join)**.** *Let* r, s *be two relations,* $Scope(\{r, s\}) = \{x_{k_1}, \ldots, x_{k_m}\}$ *be their joint scope and* $\mathcal{D}_{r \otimes s}$ *their joint domain space. The combination of* r *and* s *(noted* $r \otimes s$*) is a utility relation over* $Scope(\{r, s\})$ *such that* $(r \otimes s)(d) = r(d_r) + s(d_s)$ *for all* $d \in \mathcal{D}_{r \otimes s}$*, where* $d_r \in \mathcal{D}_r$ *and* $d_s \in \mathcal{D}_s$ *are the projections of* d *over the domains of relations* r *and* s *respectively. Ex:* $(r_{12} \otimes r_{24})(0, 1, 0) = r_{12}(0, 1) + r_{24}(1, 0)$*. Thus, the join operator is a combination operator that joins the knowledge represented by two relations into a single one by adding their values.*

Definition 8 (Summarization)**.** *The summarization operator of relation* r *over a set of variables* V *returns a relation over* V *such that* $(\bigoplus_V r)(d_V) = \max_{d \in \mathcal{D}_{\bar{V}^r}} r(d_V, d)$.

Ex:$(\bigoplus_{\{x_2\}} r_{12})(0) = \max_{d_1 \in \mathcal{D}_1} r_{12}(d_1, 0)$ *and* $(\bigoplus_{\{x_2\}} r_{12})(1) = \max_{d_1 \in \mathcal{D}_1} r_{12}(d_1, 1)$. *Notice that we can employ the summarization operator by specifying the variables to eliminate from a relation as follows* $\bigoplus_{\setminus V} r = \bigoplus_{\bar{V}^r} r = \bigoplus_{Scope(r)\setminus V} r$. *The summarization operator sums up the utility that a relation contains over a set of variables.*

Definition 9 (Slice)**.** *The slice of a relation r by an assignment σ over V is a utility relation over $\mathcal{D}_{\bar{V}^r}$ such that $(\underset{\sigma}{\triangledown} r)(d_{\bar{V}^r}) = r(d_V, d_{\bar{V}^r})$ where $d_V \in \mathcal{D}_V$ contains the values set by σ to the variables in V. Ex: $V = \{x_2\}$, $\sigma(x_2) = 1$ then $(\underset{\sigma}{\triangledown} r_{12})(0) = r_{12}(0, 1)$ and $(\underset{\sigma}{\triangledown} r_{12})(1) = r_{12}(1, 1)$.*

4.2 Background: The Generalized Distributive Law

GDL (Aji and McEliece, 2000) is a general message-passing algorithm that exploits the way a global function factors into a combination of local functions to compute the objective function in an efficient manner. The importance of the GDL framework stems from unifying a family of techniques (e.g. Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms to name a few), which have been widely used in different areas for a wide range of applications (e.g. by information theory to decode error correcting codes (MacKay, 2003) and by probabilistic inference to find the maximum a posteriori assignment in Markov Random Fields (Jensen and Jensen, 1994)).

GDL is defined over two binary operations that form a commutative semi-ring. In our case, since we are concerned with the problem of maximizing a utility function, such operations are addition and maximization.

The GDL algorithm has different versions, depending on the problem representation used. When applied directly over the global objective function, GDL is an iteratively and approximate algorithm, also known as Max-Sum or Loopy Belief Propagation. In contrast, when the objective function is compiled into a *junction tree* (Jensen and Jensen, 1994) GDL is a complete algorithm that ensures optimality and convergence. In this chapter we focus on the second case.

For this reason, before describing GDL operation, we start with an overview of junction trees in the next section.

4.2.1 Junction trees

In this section we introduce junction trees. For extended reviews on junction trees, refer to (Jensen and Jensen, 1994; Bishop, 2007).

Definition 10. *A **junction tree (JT)** is a tree of cliques that can be represented as a tuple $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ where:*

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ *is a set of variables.*

- $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ *is a set of cliques, where each clique* $\mathcal{C}_i$ *is a subset of variables* $\mathcal{C}_i \subseteq \mathcal{X}$
- $\mathcal{S}$ *is a set of separators, where each separator* s_{ij} *is an edge between clique* $\mathcal{C}_i$ *and* $\mathcal{C}_j$ *containing their intersection, namely* $s_{ij} = \mathcal{C}_i \cap \mathcal{C}_j$*; and*
- $\Psi = \{\psi_1, \ldots, \psi_m\}$ *is a set of potentials, one per clique, where potential* ψ_i *is a function assigned to clique* $\mathcal{C}_i$ *.*

Furthermore, the following properties must hold:

- ***Single-connectedness.*** *Separators create exactly one path between each pair of cliques.*
- ***Covering.*** *Each potential domain is a subset of the clique to which it is assigned, namely* $Scope(\psi_i) \subseteq \mathcal{C}_i$*.*
- ***Running intersection.*** *If a variable* x_i *is in two cliques* $\mathcal{C}_i$ *and* $\mathcal{C}_j$*, then it must also be in all cliques on the (unique) path between* $\mathcal{C}_i$ *and* $\mathcal{C}_j$*.*

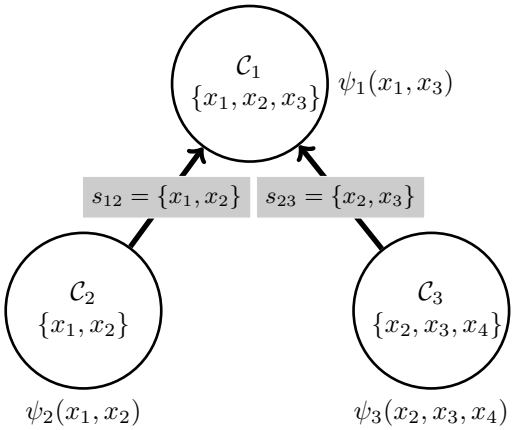


Figure 4.1: Example of junction tree.

Example 4.2.1.1 (junction tree)**.** *Figure 4.1 shows a junction tree whose circles stand for cliques, labelled with the variables each one contains, and edges (between cliques) stand for separators. Thus, for example,* $\mathcal{C}_1$ *contains variables* x_1, x_2, x_3*;* $\mathcal{C}_3$ *contains variables* x_2, x_3, x_4*; and their separator is composed of their intersection, namely variables* x_2 *and* x_3*. Observe that the covering property holds. For example, clique* $\mathcal{C}_2$ *contains variables* x_1, x_2 *because the domain of its potential,* Ψ_2*, is composed of these variables. Notice also that the running intersection property holds. For example, variable* x_2*, which is included in cliques* $\mathcal{C}_2$ *and* $\mathcal{C}_3$*, is also in* $\mathcal{C}_1$*, which is on the path between them.*

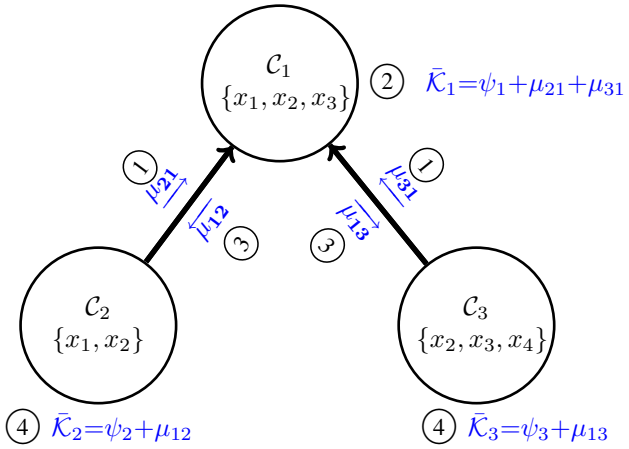


Figure 4.2: Messages exchanged and operations performed during the GDL execution over the junction tree of figure 4.1.

Likewise variables in DCOP, we assume that the variables in a JT are defined over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Moreover, $\mathcal{D}_{\mathcal{C}_i}$ stands for de domain of clique $\mathcal{C}_i$, namely the joint domain space of $\mathcal{C}_i$' variables.

Now, we proceed in the next section to detail the operation of GDL algorithm over junction trees, as the one exemplified in figure 4.1.

4.2.2 GDL operation

The purpose of GDL is that cliques distributedly compute some objective function that is factored among them. With that goal GDL defines a message-passing phase for cliques to exchange information about their variables. To illustrate the way GDL operates, consider the following example. Say that our goal is to distributedly maximise some objective function $f(x_1, x_2, x_3, x_4) = \psi_1(x_1, x_3) + \psi_2(x_1, x_2) + \psi_3(x_2, x_3, x_4)$, such that ψ_1, ψ_2 and ψ_3 are set as potentials in the directed junction tree of figure 4.1. Then, since the junction tree is directed, messages are sent in two different message-passing phases:

(i) *up the tree* so that each clique sends a message to its clique parent after receiving messages from all its children; and

(ii) *down the tree* so that each clique sends a message to its children after receiving a message from its parent.

Example 4.2.2.1 (GDL execution)**.** *Table 4.1 shows a trace of the operation of GDL over the junction tree in figure 4.1 specifying the operations realised at each step. Figure 4.2 depicts the same execution by drawing the messages and operations over the junction tree, where circled numbers stand for steps. At step 1,* $\mathcal{C}_3 = \{x_2, x_3, x_4\}$ *sends*

Step	Message/local knowledge ($\bar{\mathcal{K}}$)
1.	$\mu_{21}(x_1, x_2) = \bigoplus_{\{x_1,x_2\}} \psi_2(x_1, x_2)$
1.	$\mu_{31}(x_2, x_3) = \bigoplus_{\{x_2,x_3\}} \psi_3(x_2, x_3, x_4)$
2.	$\bar{\mathcal{K}}_1(x_1, x_2, x_3) = \psi_1(x_1, x_3) \otimes \mu_{21}(x_1, x_2) \otimes \mu_{31}(x_2, x_3)$
3.	$\mu_{12}(x_1, x_2) = \bigoplus_{\{x_1,x_2\}} [\psi_1(x_1, x_3) \otimes \mu_{31}(x_2, x_3)]$
3.	$\mu_{13}(x_2, x_3) = \bigoplus_{\{x_2,x_3\}} [\psi_1(x_1, x_3) \otimes \mu_{21}(x_1, x_2)]$
4.	$\bar{\mathcal{K}}_2(x_1, x_2) = \psi_2(x_1, x_2) \otimes \mu_{12}(x_1, x_2)$
4.	$\bar{\mathcal{K}}_3(x_2, x_3, x_4) = \psi_3(x_2, x_3, x_4) \otimes \mu_{13}(x_2, x_3)$

Table 4.1: Trace of GDL over the junction tree of figure 4.1.

a message μ_{31} to $\mathcal{C}_1 = \{x_1, x_2, x_3\}$ with the values of its local function, ψ_3, after 'filtering out' the variables that the cliques do not share in the separator, namely x_4. At step 1 also, $\mathcal{C}_2$ performs an analogous operation to send a message μ_{21} to $\mathcal{C}_1$. At step 2, after $\mathcal{C}_1$ receives its children's local functions for its variables (x_1, x_3), it combines them into $\bar{\mathcal{K}}_1$. $\bar{\mathcal{K}}_1$ is a function that stands for $\mathcal{C}_1$ knowledge over its variables. At that point, since $\mathcal{C}_1$ has received messages from all its children, $\bar{\mathcal{K}}_1$ has complete knowledge about its variables. Then, at step 3, $\mathcal{C}_1$ sends messages back to its children that contain the combination (join operation) of its local function, ψ_1, with other children messages. Thus, $\mathcal{C}_2$ receives a message from $\mathcal{C}_1$ that contains the potential ψ_1 combined with μ_{31}, whereas $\mathcal{C}_3$ receives the potential ψ_1 combined with μ_{21}. Finally, at step 4, $\mathcal{C}_2$ can compute $\bar{\mathcal{K}}_2$ while $\mathcal{C}_3$ can compute $\bar{\mathcal{K}}_3$.

Once the message-passing phase of GDL is over, each clique $\mathcal{C}_i$ has complete knowledge of the global objective function for the variables in the clique, encoded by means of function $\bar{\mathcal{K}}_i$. Now each clique can locally compute the assignment of variables that maximises the objective function. However, such local computations do not guarantee that cliques agree on their assignments. This is the case when several assignments maximise the objective function.

Formal description

In what follows we generalise the example above to provide a more formal description of GDL that will help formalising the operations that Action-GDL requires. For the sake of simplicity, we restrict our description to the max-sum commutative semi-ring. Notice though that GDL applies to a larger variety of semi-rings. We refer the interested reader to (Aji and McEliece, 2000) for an excellent discussion on this issue.

Firstly, the knowledge of a clique $\mathcal{C}_i$ results from the combination of its local function with each of the messages received from its neighbours. For instance, in table 4.1 (step 2) the knowledge of clique $\mathcal{C}_1$ is assessed as the combination of its potential, Ψ_1, with the messages received from its neighbours $\mathcal{C}_2$ and $\mathcal{C}_3$. More formally, the knowledge of a clique $\mathcal{C}_i$ is defined as a knowledge relation $\bar{\mathcal{K}}_i : \mathcal{D}_{\mathcal{C}_i} \rightarrow \mathbb{R}$. Initially, $\bar{\mathcal{K}}_i$ is set

to be the local potential ψ_i, but when $\bar{\mathcal{K}}_i$ is updated, GDL uses the following rule:

$$\bar{\mathcal{K}}_i = \psi_i \otimes \left[\bigotimes_{\mathcal{C}_k \in N(\mathcal{C}_i)} \mu_{ki} \right] \tag{4.1}$$

where $N(\mathcal{C}_i)$ is the set of adjacent cliques to $\mathcal{C}_i$ in the junction tree.

Secondly, notice that if two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ are connected by a separator s_{ij}, the message from $\mathcal{C}_i$ to $\mathcal{C}_j$ is an utility message μ_{ij} over variables in s_{ij}. Initially all μ_{ij} messages are set to 0. For instance, in table 4.1 (step 3) the message μ_{13} that clique $\mathcal{C}_1$ exchanges with $\mathcal{C}_3$ is a relation over the variables in separator $s_{13} = \{x_2, x_3\}$. When a clique $\mathcal{C}_i$ sends a message to $\mathcal{C}_j$, it combines its local knowledge with all messages it has received from its neighbours other than $\mathcal{C}_j$, and transmits the result to $\mathcal{C}_j$. Thus, $\mathcal{C}_1$ assesses μ_{13} as the combination of its potential, Ψ_1, with the message received from $\mathcal{C}_2$. Following (Aji and McEliece, 2000), we can regard a junction tree as a communication network where an edge from $\mathcal{C}_i$ and $\mathcal{C}_j$ is a transmission line that "filters out" dependence (by summarization in our case) on all variables but those common to $\mathcal{C}_i$ and $\mathcal{C}_j$. When a message μ_{ij} is updated, the following rule applies:

$$\mu_{ij} = \bigoplus_{s_{ij}} \left[\psi_i \otimes \bigotimes_{\mathcal{C}_k \in N(\mathcal{C}_i)} \mu_{ki} \right] \tag{4.2}$$

To summarise, equations 4.1 and 4.2 are the basis of GDL. While equation 4.2 helps a clique assess messages to send to its neighbours, equation 4.1 allows to compute the objective function at a particular clique.

Consider two special cases regarding the application of GDL (Aji and McEliece, 2000): the *single-vertex* problem, when the goal is to compute the objective function at a single clique, and the *all-vertices* problem, when the goal is to compute the objective function at all cliques. For instance, consider the trace in figure 4.1. At step 3, the single-vertex problem for clique $\mathcal{C}_1$ is solved, whereas at step 4 the all-vertices problem is solved because all cliques can assess the objective function.

As mentioned above, GDL is a synthesis of the work in many fields. More concretely, it generalises Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms among others. In the next section, we observe that it is also the case of the Cluster Tree Elimination (CTE) algorithm described in (Dechter, 2003), a message-passing algorithm that can help solve several automated reasoning tasks over a tree decomposition.

4.2.3 Relationship with the Cluster Tree Elimination algorithm

Now we turn our attention to the Cluster Tree Elimination (CTE) algorithm described in (Dechter, 2003) to compare it with GDL. CTE is a message-passing algorithm that can help solve several automated reasoning tasks (e.g. constraint satisfaction, most probable explanation in a belief network, etc) over a tree decomposition. It is a complete algorithm, with exponential complexity in time and space (Kask et al., 2005).

The operation of CTE is the same as GDL to solve the all-vertices problem, namely to assess the objective function at each clique. In fact, CTE employs equations 4.1 and 4.2 to assess the knowledge in a cluster and the messages to send between cliques respectively. More concretely, the original description of CTE is equivalent to the *fully serial* version of GDL (Aji and McEliece, 2000): a clique sends a message to a neighbour when, for the first time, it has received messages from all of its other neighbours, and computes its knowledge when, for the first time, it has received messages from all its neighbours.

4.3 The Action-GDL Algorithm

In this section we introduce Action-GDL, a specialization of the GDL algorithm in (Aji and McEliece, 2000) to efficiently solve DCOPs. Firstly, in section 4.3.1, we show how to extend GDL to efficiently solve DCOPs, namely to allow individual agents to calculate the variable assignment that maximizes the DCOP objective function of equation 2.1. Next, section 4.3.2 details a formal analysis of the complexity of Action-GDL. Finally, in section 4.3.3, we define an algorithm to distributedly compile a DCOP into a junction tree.

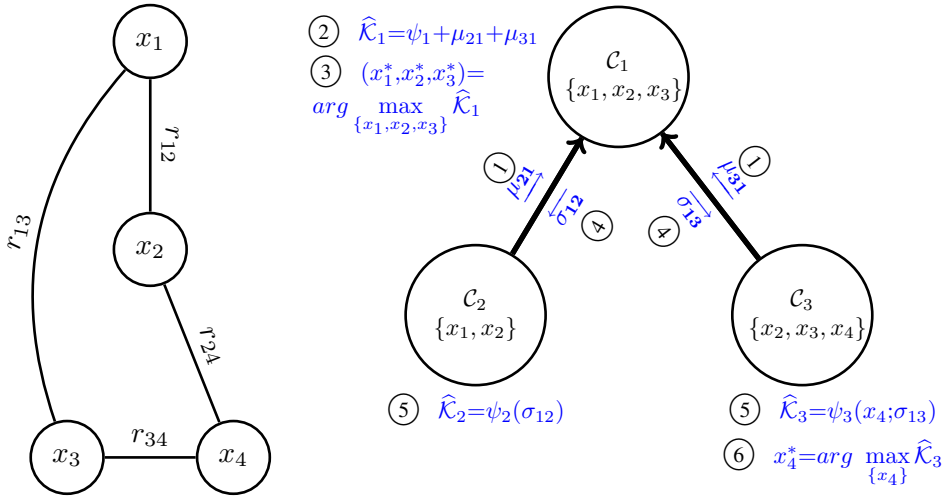
4.3.1 Extending GDL to solve DCOPs

Recall that our goal is to efficiently solve DCOPs as formalised in section 2.1. Therefore, the capability of distributedly computing an objective function,[1] as provided by GDL, is not enough. We need to go one step beyond GDL to allow agents to individually assign values to local variables such that these values are a joint assignment that maximizes the DCOP objective function. At this aim, Action-GDL extends GDL to efficiently infer such optimal assignment.

Consider a DCOP whose objective function is compiled into a junction tree. For example, the junction tree of figure 4.1 encodes the objective function of the DCOP represented in figure 4.3(a) by setting the set of potentials $\Psi = \{\psi_1, \psi_2, \psi_3\}$ as $\psi_1 = r_{13}$, $\psi_2 = r_{12}$, and $\psi_3 = r_{23} \otimes r_{34}$.

According to the description of GDL above, when a clique has received messages from all its neighbors, it counts on all the information related to its variables. Thus, it can compute its objective function. Thereafter, the clique would be able to find the optimal assignment for its variables provided that it is aware of its parent clique decisions (variables' assignments) and can set the rest of clique's variables according to them. Once the clique finds the optimal assignment for its variables, it can directly propagate its assignment down the tree. Notice that there is no need for propagating utility messages down the tree (unlike GDL when solving the all-vertices problem) because everything that is required for a child to find its optimal assignment is its parent's assignments. Therefore, when solving a DCOP, after the first message-passing phase of GDL (up the tree) is over, the second message-passing phase of GDL (down the tree) is no longer necessary. Instead, we require a second message-passing phase for

[1] In fact, each clique ends up in GDL with a summarization of the global objective function over its variables.



(a) DCOP constraint graph (b) Action-GDL execution when solving DCOP (a).

Figure 4.3: Example of (a) DCOP constraint graph; and (b) the execution of Action-GDL over the junction tree of figure 4.1 when encoding (a).

cliques to exchange *value assignments* down the tree, which is precisely the extension that Action-GDL introduces. Therefore, Action-GDL runs two phases:

(i) a *utility propagation phase* in which each agent exchanges utility messages for each of its cliques up the tree; and

(ii) a *value propagation phase* in which each agent exchanges value messages for each of its cliques down the tree.

Unlike GDL, messages exchanged down the tree are not utility messages. This leads to savings in communication, since utility messages are space-exponential in the separator size, whereas the size of a value messages is linear. Moreover, it is relevant to notice that the value propagation phase ensures that whenever multiple optimal joint decisions are feasible, cliques converge to the very same joint decision, namely to the very same solution of a DCOP.

Example 4.3.1.1 (Action-GDL execution)**.** *Table 4.2 displays a trace of Action-GDL operations over the junction tree in figure 4.1. Figure 4.3(b) depicts the same execution by drawing the messages and operations over the junction tree, where circled numbers stand for operations. Observe that the operations performed during the utility propagation phase (steps 1-2) are equivalent to the operations performed in the first phase of GDL (steps 1-2). In contrast, at step 3 the root clique $\mathcal{C}_1$ initiates the value propagation phase and assesses the optimal value for x_1, x_2 and x_3 differing from that point on from the GDL execution. At step 4, $\mathcal{C}_1$ propagates its optimal values down the tree through value messages to cliques $\mathcal{C}_2$ and $\mathcal{C}_3$. At step 5, $\mathcal{C}_2$ and $\mathcal{C}_3$ receive the values of $\{x_1, x_2\}$*

Step	Messages/local knowledge $\widehat{\mathcal{K}}$
1.	$\mu_{21}(x_1, x_2) = \bigoplus_{\{x_1,x_2\}} \psi_2(x_1, x_2)$
1.	$\mu_{31}(x_2, x_3) = \bigoplus_{\{x_2,x_3\}} \psi_3(x_2, x_3, x_4)$
2.	$\widehat{\mathcal{K}}_1(x_1, x_2, x_3) = \psi_1(x_1, x_3) \otimes \mu_{21}(x_1, x_2) \otimes \mu_{31}(x_2, x_3)$
3.	$(x_1^*, x_2^*, x_3^*) = arg\ \max_{\{x_1,x_2,x_3\}} \widehat{\mathcal{K}}_1$
4.	$\sigma_{12}(x_1, x_2) = (x_1^*, x_2^*)$
4.	$\sigma_{13}(x_2, x_3) = (x_2^*, x_3^*)$
5.	$\widehat{\mathcal{K}}_2 = \underset{\sigma_{12}}{\bigtriangledown} \psi_2(x_1, x_2)$
5.	$\widehat{\mathcal{K}}_3(x_4) = \underset{\sigma_{13}}{\bigtriangledown} \psi_3(x_2, x_3, x_4)$
6.	$x_4^* = arg\ \max_{\{x_4\}} \widehat{\mathcal{K}}_3$

Table 4.2: Trace of Action-GDL over the junction tree of Fig. 4.1

and $\{x_2, x_3\}$ *respectively. At step 6,* $\mathcal{C}_3$ *infers the remaining variable* x_4 *by using its parent optimal values* (x_2^*, x_3^*).

Algorithm 1 outlines Action-GDL. For simplicity, we have assumed that each agent x_i is assigned a single clique $\mathcal{C}_i$. Given a junction tree $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$, each clique $\mathcal{C}_i$ starts with a tuple $\langle \mathcal{C}_p, Ch_i, \psi_i, S_i \rangle$, where $\mathcal{C}_p \in \mathcal{C}$ stands for its parent, $Ch_i \subseteq 2^{\mathcal{C}}$ stands for its children, and $S_i \in \mathcal{S}$ stands for the separators relating clique $\mathcal{C}_i$ to its adjacent cliques.

During the utility propagation phase (lines 1-10), cliques exchange utility messages. The initial knowledge for each clique is its potential (line 2). Each clique waits until receiving a utility message from each of its children cliques (lines 3-4). These messages contain a utility relation over the variables shared by both cliques (their separator) and are sent by children cliques. For example, in the junction tree of figure 4.3(b), clique $\mathcal{C}_1$ waits until receiving utility messages from cliques $\mathcal{C}_2$ and $\mathcal{C}_3$, containing relations over variables in their respective separators, namely $\{x_1, x_2\}$ and $\{x_2, x_3\}$. Every time a clique receives a new utility message, it incorporates it (by using the join operator) to its local knowledge (line 5). After combining utility messages from all the children of a clique, if the clique has a parent (line 7), it summarizes that part of its local knowledge (using the summarization operator) that is of interest to its parent (by means of a utility relation over its separator). Thus, clique $\mathcal{C}_3$ summarizes its potential over variables $\{x_2, x_3\}$ (filtering out x_4), which are the variables of interest to its clique parent $\mathcal{C}_1$. After that, the result of the summarisation is sent to its parent (line 9).

During the value propagation phase (lines 11-22), cliques compute the optimal values for their variables and exchange value messages, namely decisions. Given a clique, it waits until receiving a value message (containing assignments) for all variables in common (in the separator) with its parent (line 12-14). Thus, in the junction tree of figure 4.3(b), cliques $\mathcal{C}_2$ and $\mathcal{C}_3$ wait until receiving a value message from its parent $\mathcal{C}_1$ containing the optimal values for variables in their respective separators, namely x_1^*, x_2^* and x_2^*, x_3^*. At that point, the clique has received all the knowledge, in form of utility (from children) and value (from the parent) messages, required to compute the objec-

Algorithm 1 Action-GDL($\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$)

Each clique $\mathcal{C}_i$ starts with $\langle \mathcal{C}_p, Ch_i, \psi_i, \mathcal{S}_i \rangle$ and runs:

1: **Phase I: UTILITY Propagation**
2: $\widehat{\mathcal{K}}_i = \psi_i$;
3: **for all** $\mathcal{C}_j \in Ch_i$ **do**
4: Wait for utility message μ_{ji} from $\mathcal{C}_j$
5: $\widehat{\mathcal{K}}_i = \widehat{\mathcal{K}}_i \otimes \mu_{ji}$;
6: **end for**
7: **if** $\mathcal{C}_i$ is not the tree's root **then**
8: Let $s_{ip} \in \mathcal{S}_i$ be the separator between $\mathcal{C}_i$ and its parent C_p
9: Send $\mu_{ip} = \bigoplus_{s_{ip}} \widehat{\mathcal{K}}_i$ to $\mathcal{C}_p$
10: **end if**
11: **Phase II: VALUE propagation**
12: **if** $\mathcal{C}_i$ is not the tree's root **then**
13: Wait for a value message σ_{pi} from $\mathcal{C}_p$
14: $\widehat{\mathcal{K}}_i = \underset{\sigma_{pi}}{\triangledown} \widehat{\mathcal{K}}_i$; /*Slice $\widehat{\mathcal{K}}_i$ with the value message*/
15: **end if**
16: $x_i^* = arg \max_{d \in \mathcal{D}_{Scope(\widehat{\mathcal{K}}_i)}} \widehat{\mathcal{K}}_i(d)$; /*Assess best values for unassigned local variables*/
17: $x^* = x_i^* \cup \sigma_{pi}$; /*Put together the assessed value and the message received*/
18: **for all** $\mathcal{C}_j \in Ch_i$ **do**
19: Let $\sigma_{ij} = x^*_{s_{ij}}$; /*Project over separator*/
20: Send σ_{ij} to $\mathcal{C}_j$
21: **end for**
22: **return** d^*;

tive function related to its variables. The clique slices its knowledge by incorporating the already inferred decisions (line 15), and computes the optimal values for the rest of its variables (line 17). For instance, clique $\mathcal{C}_3$ slices its knowledge by incorporating the values x_2^*, x_3^* inferred by $\mathcal{C}_1$, and computes the optimal value of its only remaining variable x_4. Once a clique finds the optimal assignment for its variables, it can propagate it down the tree (lines 19-22). Notice however that a clique only propagates variable assignments required by its children cliques, namely assignments for variables in their separator.

To summarize, according to algorithm 1, the knowledge of a clique $\mathcal{C}_i$ at the end of an Action-GDL run is:

$$\widehat{\mathcal{K}}_i = \underset{\sigma_{pi}}{\triangledown} \left[\psi_i \otimes \bigotimes_{\mathcal{C}_j \in Ch_i} \mu_{ji} \right] \tag{4.3}$$

Observe that in contrast with GDL, which computes the knowledge of a clique as the combination of utility messages received from all neighbours (see equation 4.3), the

knowledge of a clique in Action-GDL contains only the combination of utility messages from children whereas the value message from the clique parent is used to slice this knowledge by fixing the values of the already inferred variables.

4.3.2 Computation and communication complexity

In this section, we discuss the computational and communication complexity of Action-GDL (Algorithm 1). As we show next, we can readily assess Action-GDL complexity from cliques' and separators' sizes of a junction tree.

Communication complexity

Communication complexity is measured in terms of the number and the size of the messages exchanged during the execution of the algorithm.

Action-GDL requires a number of messages linear in the number of edges in the junction tree because agents exchange one value message and one utility message per separator. The communication complexity lies in the size of utility messages, which is the number of variables of the separator ($size(\mu_{ij}) \in O(\prod_{x_k \in s_{ij}} |\mathcal{D}_k|)$), because the size of value messages is linear ($size(\sigma_{ij}) \in O(\sum_{x_k \in s_{ij}} log(|\mathcal{D}_k|))$).

Computation complexity

Computation complexity is measured in terms of memory space and number of operations required during the execution of the algorithm.

The local memory required by each clique $\mathcal{C}_i$ depends on the size of the received messages, which for utility messages is exponential to the number of variables in the separators, and on the size of its local knowledge $\mathcal{K}_i$, which is exponential to the number of variables in the clique. Regarding the computation required by each clique to build messages and find assignments, it also scales with the number of variables in the clique. In more detail, the operations required by a clique $\mathcal{C}_i$ are the following:

- the combination of children messages into the local knowledge (lines 3-6) requires $\prod_{x_k \in \mathcal{C}_i} |\mathcal{D}_k|$ operations, and hence, is exponential to the number of variables in the clique.
- the summarization of the local knowledge over a separator to compute a utility message (line 9) requires $\prod_{x_k \in \mathcal{C}_i} |\mathcal{D}_k|$ operations, and hence, it is exponential to the number of variables in the clique.
- the slice of knowledge to incorporate optimal assignments from its parent along with assessment of the remaining variables (lines 12-16) require $\prod_{x_k \in \mathcal{C}_i \setminus s_{ip}} |\mathcal{D}_k|$ operations[2] and hence, it is bounded by the number of variables in the clique.

Therefore, the communication and computation complexity of Action-GDL are dominated by the size of the largest clique and the larger separator in the junction tree. Moreover, because it is known that the size of the largest clique in the best junction tree

[2] s_{ip} stands for the parent's separators

is equal to the treewidth (plus one) (Jensen and Jensen, 1994; Bishop, 2007), the computational complexity of Action-GDL is also bounded by the treewidth of the DCOP constraint graph.

4.3.3 Distributed Junction Tree Generator

As explained in section 4.3.1 when detailing Action-GDL operation, Action-GDL runs over a junction tree where cliques are assigned to agents. Then, we shall refer to such junction tree as Distributed Junction Tree (DJT). In a Distributed Junction Tree each agent a_i is assigned cliques for its variables whose potential is composed of relations that a_i knows. Recall that, as defined in chapter 2, in a DCOP the relations known by an agent are those that include any of its variables.

For example consider the DCOP in figure 4.3(a) with the following assignment of variables to agents: x_1 to a_1, x_2 to a_2 and x_3, x_4 to a_3. When we compiled the objective function of this DCOP into the junction tree of figure 4.3(b) we created three cliques, one for each agent. Additionally, the potential of each clique $\mathcal{C}_i$ is composed only of relations that the owner agent a_i knows. Thus, ψ_1 is composed of r_{13} known by a_1 as owner of x_1, ψ_2 of r_{12} known by a_2 as owner of x_2 and ψ_3 is composed of the combination of r_{34} and r_{24} known by a_3 as owner of x_3,x_4. Hence the junction tree of figure 4.3(b) is a distributed junction tree for the DCOP of figure 4.3(a).

However, that is not always the case. Indeed, it has been argued Petcu (2007) that traditional triangulation-based methods to compile junction trees[3], are not suitable when applied to problems that are distributed by nature because they produce junction trees disregarding the structure of the problem. Following any of these methods, we can create a clique $\mathcal{C}_i$ with a potential ψ_i composed of the combination of r_{13}, r_{34} and r_{24}. Thus, in this case, ψ_i contains a set of relations that are not known by any single agent in the DCOP and hence, it can not be included in any distributed junction tree.

To overcome this limitation, here we propose an alternative algorithm, the Distributed Junction Tree Generator (DJTG), which allows agents to compile a DCOP into distributed junction tree by exchanging a linear number of messages. The resultant distributed junction tree can be readily fed into, and hence solved by, Action-GDL. The DJTG algorithm builds on an existing distributed method for building a junction tree Paskin et al. (2005) that, unlike traditional triangulation methods, is based on directly ensuring the running intersection property (RIP) over the problem. Recall that the RIP (defined in section 10) ensures that if a variable x_i is in two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ then it is also in all the cliques in the path between them. The DJTG algorithm redefines this method in the context of DCOPs.

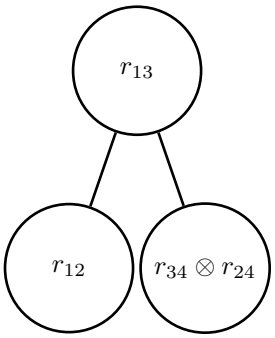
The DJTG algorithm captures the distribution of a DCOP because each agent:

(i) creates its own cliques related to its variables, leading to an explicitly relationship agent-to-clique; and

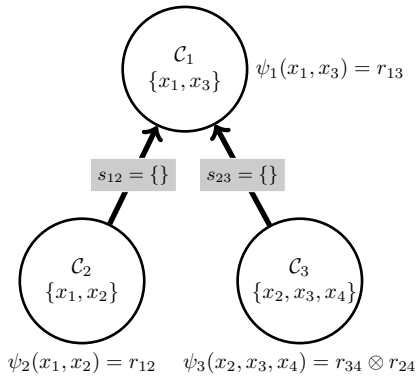
(ii) restricts the potential of its clique to the relation that it knows.

The DJTG algorithm receives as input a set of relations distributed among agents and a spanning tree. Figure 4.4(a) shows a spanning tree over the set of relations of

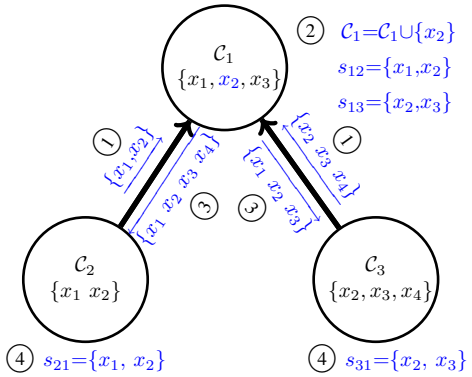
[3]We refer to triangulation methods based on the one proposed in Jensen and Jensen (1994)



(a) Distribution of relations the DCOP in figure 4.3(a) and spanning tree (input of DJTG).



(b) DJTG execution (preprocessing phase).



(c) DJTG execution (RIP phase).

Figure 4.4: Example of DJTG execution.

the DCOP constraint graph of figure 4.3(a). It considers the following assignment of variables to agents: x_1 to a_1, x_2 to a_2 and x_3, x_4 to a_3. Observe that r_{13} is assigned to a_1 and linked to r_{12}, assigned to agent a_2; and to the combination of r_{34} and r_{24}, assigned to a_3.

Formally, the input of the DJTG algorithm is a tuple $\langle \mathcal{A}, \mathcal{X}, R, \kappa, ST \rangle$, where: $\mathcal{A}$ is a set of agents; $\mathcal{X}$ is a set of variables; R is a set of relations; κ is a function that maps agents to a set of relations; and ST a spanning tree defined over the relations in R.

Then, the DJTG algorithm runs two phases:

(i) a *pre-processing phase* where agents an initial representation; followed by

(ii) a *message-passing phase* that calculates the unique set of minimal cliques that satisfy the RIP.

During the pre-processing phase, each agent a_i creates a clique $\mathcal{C}_i$ for each of its as-

Step	DJTG Trace	Step	DJTG Trace
P.	$\Psi_1 = r_{13}, \mathcal{C}_1 = Scope(\Psi_1)$	2.	$s_{12} = \mathcal{C}_1 \cap \varphi_{21}$
P.	$\Psi_2 = r_{12}, \mathcal{C}_2 = Scope(\Psi_2)$	2.	$s_{13} = \mathcal{C}_1 \cap \varphi_{31}$
P.	$\Psi_3 = r_{34} \otimes r_{24}, \mathcal{C}_2 = Scope(\Psi_3)$	3.	$\varphi_{12} = Scope(\Psi_1) \cup \varphi_{31}$
1.	$\varphi_{21} = Scope(\Psi_1)$	3.	$\varphi_{13} = Scope(\Psi_1) \cup \varphi_{21}$
1.	$\varphi_{31} = Scope(\Psi_3)$	4.	$s_{21} = \mathcal{C}_2 \cap \varphi_{12}$
2.	$\mathcal{C}_1 = \mathcal{C}_1 \cup \{\varphi_{21} \cap \varphi_{31}\}$	4.	$s_{31} = \mathcal{C}_3 \cap \varphi_{13}$

Table 4.3: Trace of DJTG over the junction tree of figure 4.1.

signed relations , $r \in \kappa(a_i)$, setting the clique's potential ψ_i to r. Figure 4.4(b) shows the initial junction tree structure produced during the DJTG preprocessing phase over the input depicted in figure 4.4(a). Agent a_1 creates clique $\mathcal{C}_1$, agent a_2 creates clique $\mathcal{C}_2$, and agent a_3 creates clique $\mathcal{C}_3$. Table 4.3 details the operations performed during this pre-processing phase (tagged as step $P.$). Cliques are initially set to their potential domain, namely $\mathcal{C}_i = Scope(\psi_i)$, in order to readily ensure the covering property. Thus, clique $\mathcal{C}_1$ is initially set to $\{x_1, x_3\}$, which are the variables in the scope of its potential, uniquely composed of relation r_{13}.

Moreover, for every two relations r_{V_i}, r_{V_j} connected in the $\mathcal{ST}$, agents create a separator s_{ij} linking their corresponding cliques $\mathcal{C}_i$ and $\mathcal{C}_j$. Thus, in figure 4.4(b), cliques $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected as their relations r_{13} and r_{12} in the $\mathcal{ST}$. Notice that the structure shown in figure 4.4(b) does not satisfy the RIP property: variable x_2 is included in $\mathcal{C}_2$ and $\mathcal{C}_3$ but not in $\mathcal{C}_1$ which is in the path between them.

The second phase of DJTG is responsible for ensuring the RIP. During that phase, each agent exchanges for each one of its cliques, $\mathcal{C}_i$, *reachability* messages with agents related to $\mathcal{C}_i$'s neighbours that contain the set of reachable variables from $\mathcal{C}_i$. Figure 4.4(c) shows the messages exchanged, with the set of reachable variables, and the operations performed during this stage, where circled numbers stand for the execution step. Table 4.3 details the operations performed at each step of the execution.

The set of reachable variables from a clique $\mathcal{C}_i$ to $\mathcal{C}_j$, namely φ_{ij}, is calculated as the union of: (i) $\mathcal{C}_i$'s potential domain; and (ii) the variables reachable from $\mathcal{C}_i$'s neighbours other than $\mathcal{C}_j$'s. Thus, at step 1, agent a_2 sends a message to agent a_1 for clique $\mathcal{C}_2$ that contains its potential domain $\{x_1, x_2\}$, namely the variables that can be reached from clique $\mathcal{C}_2$. At step 3, agent a_1 sends a message to agent a_2 for clique $\mathcal{C}_1$ that contains variables $\{x_1, x_2, x_3, x_4\}$, namely the variables that can be reached from clique $\mathcal{C}_1$. These variables are the result of the union of $\mathcal{C}_1$'s potential domain, namely (x_1, x_3), with the reachable variables from $\mathcal{C}_3$, namely $\{x_2, x_3, x_4\}$. Formally, a reachability message φ_{ij} from $\mathcal{C}_i$ to $\mathcal{C}_j$ is assessed as:

$$\varphi_{ij} = Scope(\Psi_i) \cup \left[\bigcup_{\substack{\mathcal{C}_k \in N(\mathcal{C}_i) \\ k \neq j}} \varphi_{ki} \right] \tag{4.4}$$

Once an agent receives, for a given clique, *reachability* messages from all its neighbours, it redefines its clique by adding variables that are in more than one *reachability*

message and assesses the separators with its neighbours. Formally, upon reception of all reachability messages for a clique $\mathcal{C}_i$, its new variables can be computed as follows:

$$\mathcal{C}_i = \mathcal{C}_i \cup \left[\bigcup_{\substack{\mathcal{C}_j, \mathcal{C}_k \in N(\mathcal{C}_i) \\ j \neq k}} \varphi_{ji} \cap \varphi_{ki} \right] \tag{4.5}$$

For instance, in figure 4.4(c) agent a_1 receives two reachability messages for clique $\mathcal{C}_1$: one with $\{x_1, x_2\}$ from clique $\mathcal{C}_2$ associated to a_2; and another one with $\{x_2, x_3, x_4\}$ from clique $\mathcal{C}_3$ associated to a_3. Since both messages contain x_2, agent a_1 redefines $\mathcal{C}_1$ to also include x_2.

After computing cliques, it is straightforward for agents to assess separators (see definition 10). Thus, an agent a_i computes the separator between its clique $\mathcal{C}_i$ and its neighbour $\mathcal{C}_j \in N(\mathcal{C}_i)$ as the intersection between $\mathcal{C}_i$ and the reachability message received from $\mathcal{C}_j$, namely $s_{ij} = \mathcal{C}_i \cap \varphi_{ji}$. At step 4, in figure 4.4(c), agent a_3 assesses the separator between its clique $\mathcal{C}_3$ and its neighbor $\mathcal{C}_1$ as $\{x_2, x_3\}$. Variables in this separator are the results of the intersection of $\mathcal{C}_3$, namely $\{x_2, x_3, x_4\}$, with the reachable variables from $\mathcal{C}_1$, namely $\{x_1, x_2, x_3\}$.

As a result, the final junction tree built by agents in figure 4.4(c) is equivalent to the valid junction tree in figure 4.3(b), satisfying the RIP. Hence, we have shown that from a DCOP, agents can distributedly compute a distributed junction tree using a linear number of messages.

Finding a good distributed junction tree

Given a set of n relations, there are n^{n-2} different spanning trees over them. For each one we can compile the corresponding distributed junction tree with the DJTG algorithm. It is known from Jensen and Jensen (1994) that finding the optimal junction tree is NP-hard, so it is reasonable to wonder what we can do to find good spanning trees to use as input for the DJTG algorithm. However, it turns out that existing heuristics proposed in the literature for DCOPs and distributed junction tree construction can be expressed, explicitly or implicitly, as a set of relations connected by a spanning tree that we can use as input of the DJTG. On the one hand, there are heuristics (Paskin et al., 2005) that directly assess a spanning tree defined over relations and we can readily exploit them. On the other hand, sections 4.4.1 and 4.4.2 below show how to take advantage of the heuristics proposed for pseudotrees (Petcu, 2007; Atlas and Decker, 2007) and cross-edge trees (Atlas and Decker, 2007) to create efficient distributed junction trees.

4.4 Generality of Action-GDL

In the previous section we have presented Action-GDL. In this section we show the generality of Action-GDL by showing that it unifies two state-of-the-art dynamic programming optimal DCOP algorithms that are based on GDL: DPOP (Petcu and Faltings, 2005b) and DCPOP (Atlas and Decker, 2007).

4.4.1 Action-GDL generalizes DPOP

In this section we prove that DPOP is a particular case of Action-GDL when it is executed under certain junction trees. We say that two distributed algorithms are equivalent if (i) agents perform the same computation and (ii) agents exchange the same messages. First, we give an overview of the DPOP algorithm in section 4.4.1. Next we prove that Action-GDL generalizes DPOP by: (1) providing a mapping from pseudotrees (input of DPOP) to junction trees (section 4.4.1); and (2) proving that given any pseudotree, the execution of DPOP is equivalent to the execution of Action-GDL over the junction tree produced by our mapping for the pseudotree (section 4.4.1).

Overviewing DPOP

DPOP (Petcu and Faltings, 2005b) is a complete state-of-the-art dynamic programming algorithm to solve DCOPs. DPOP compiles a DCOP in a pseudotree (PT), namely a rooted tree with the same variables as the DCOP and the property that adjacent nodes from the DCOP constraint graph fall in the same branch of the tree.

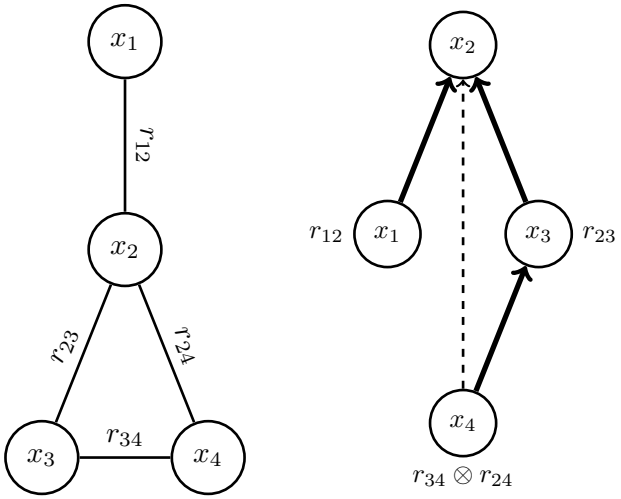
Figure 4.5(b) shows a pseudotree for the constraint graph in figure 4.5(a). A pseudotree of a constraint graph has two kinds of edges: *tree-edges* (boldfaced lines); and *pseudoedges* (dashed lines). These edges stand for two relationships between variables: (1) parent/children for variables connected through an edge (e.g. in figure 4.5(b) x_2 is the parent of x_3); (2) pseudoparent/pseudochildren for variables connected through a pseudoedge (e.g. x_4 is a pseudochild of x_2). Therefore, we can represent a pseudotree as a pair $\langle P, PP \rangle$, where P and PP are functions that map each variable to its parent and pseudoparents, respectively. We obtain functions Ch and PCh, which return a variable's children and pseudochildrens respectively, from the functions above as $Ch(x_i) = \{x_j \in \mathcal{A} | P(x_j) = x_i\}$ and $PCh(x_i) = \{x_j \in \mathcal{A} | x_i \in PP(x_j)\}$.

Thus, when running DPOP, agents start with a pre-processing phase, to generate a pseudotree by running a distributed Depth First Search (DFS) algorithm guided by some heuristic. Then, given a pseudotree, DPOP has two message-passing phases: (1) to exchange *utilities* about variables; and (2) to propagate *values* of variables inferred. Algorithm 2 shows these two phases in terms of the operators introduced in section 4.1. Such encoding will ease the comparison with Action-GDL in section 4.4.1.

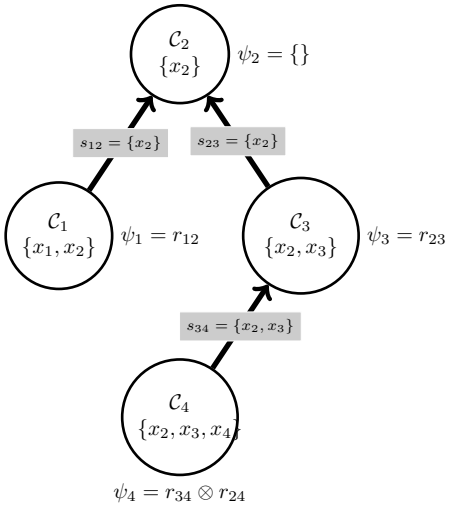
In DPOP the initial knowledge of an agent x_i, namely $\mathcal{K}_i^0$, is set to the combination of some unary relation involving x_i and of some binary relations linking x_i with one of its parent or pseudoparent variables. Thus, in figure 4.5(b) the knowledge of agent x_4 ($\mathcal{K}_4^0$) is initially composed of relations r_{24} and r_{34} (no unary relation in that case).

During the first message-passing phase, the utility propagation phase (lines 1-9), each agent x_i receives utility messages from all its children variables. The utility message that agent x_i exchanges with the agent related to its parent variable, x_p, is the summarization of its current knowledge filtering out x_i (line 8). Formally:

$$\mu_{ip} = \bigoplus_{\setminus x_i} \left[\mathcal{K}_i^0 \otimes \bigotimes_{x_j \in Ch(x_i)} \mu_{ji} \right] \tag{4.6}$$



(a) DCOP constraint graph. (b) Pseudotree PT.



(c) Junction tree $JT = \gamma(\Phi, PT)$.

Figure 4.5: Example of constraint graph, a pseudotree and its equivalent junction tree.

During the second message-passing phase, the value propagation phase (lines 10-19), each agent x_i receives a value message (σ_{pi}) from the agent assigned to its parent variable, x_p. That value message contains assignments for all variables in the domain of the utility message (μ_{ip}) that agent x_i has sent to x_p in the previous phase. Once agent x_i has received the value message from its parent x_p, agent x_i restricts its knowledge by incorporating the assigned variables (line 13).

Algorithm 2 DPOP($\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$,$\langle P, PP \rangle$)

Each agent $x_i \in \mathcal{X}$, receives $\langle P_i, PP_i, \mathcal{K}_i^0 \rangle$ where $\mathcal{K}_i^0 = r_i \otimes \bigotimes_{x_k \in \{P(x_i)\} \cup PP(x_i)} r_{ik}$ and runs:

1: **Phase I: UTILITY Propagation**
2: $\mathcal{K}_i = \mathcal{K}_i^0$;
3: **for all** $x_j \in Ch(x_i)$ **do**
4: Wait for the utility message μ_{ji} from x_j
5: $\mathcal{K}_i = \mathcal{K}_i \otimes \mu_{ji}$;
6: **end for**
7: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
8: Send $\mu_{ip} = \bigoplus_{\setminus x_i} \mathcal{K}_i$ to x_p
9: **end if**
10: **Phase II: VALUE propagation**
11: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
12: Wait for a value message σ_{pi} from x_p
13: $\mathcal{K}_i = \bigtriangledown_{\sigma_{pi}} \mathcal{K}_i$; /*Slice $\mathcal{K}_i$ with the value message*/
14: **end if**
15: $x_i^* = arg \max_{d_i \in \mathcal{D}_i} \mathcal{K}_i(d_i)$; /* Assess best value for x_i */
16: $x^* = x_i^* \cup \sigma_{pi}$; /* Put together the assessed value and the message received. */
17: **for all** $x_j \in Ch(x_i)$ **do**
18: Send $\sigma_{ij} = x^*_{Scope(\mu_{ji})}$ to x_j /* Send to x_j the variables he is interested in */
19: **end for**
20: **return** d_i^*;

Then, agent x_i assesses the value of x_i as the one that maximizes its local knowledge (line 15) and completes it with the value message received from its parent (line 16). Thereafter, agent x_i propagates to every children of x_i a value message with the values assigned to already decided variables that it is interested in (lines 17-19) [4].

To summarize, from algorithm 2 we obtain that the knowledge of agent x_i at the end of a DPOP execution is:

$$\mathcal{K}_i = \bigtriangledown_{\sigma_{pi}} \left[\mathcal{K}_i^0 \otimes \bigotimes_{x_j \in Ch(x_i)} \mu_{ji} \right] \tag{4.7}$$

Mapping pseudotrees to junction trees

Before proving the equivalence of Action-GDL and DPOP, in this section we define a mapping that builds a junction tree from a pseudotree. First of all, we offer the intuitions

[4] When looking at lines 17-19, recall that $x^*_{Scope(\mu_{ji})}$ stands for the values to be assigned to the variables in the domain of the utility message μ_{ji}.

behind our mapping. In general, we propose to map each pseudotree to a junction tree with as many cliques as nodes in the pseudotree. In fact, for each node in a pseudotree we require its counterpart as a clique in the junction tree to be produced by the mapping. Hereafter, we consider the variables to include in each clique. For each node in the pseudotree, its clique in the junction tree must contain: (1) the node's variable; (2) the variables expected by the node's parents/pseudoparents up in the pseudotree; and (3) the variables that the node's children need to forward up the pseudotree.

We will refer to the node's variable and the second set of variables as the *directly related variables* (DRV), and to the third set of variables as the *inherited related variables* (IRV). Hence, given a node x_i in a pseudotree, we can readily define the variables of its clique by wrapping up directly and inherited related variables as follows:

$$\mathcal{C}_i = DRV(x_i) \cup IRV(x_i) \tag{4.8}$$

On the one hand, the directly related variables of a node include its variable, its parents' and its pseudoparents'. Formally:

Definition 11. *Given a variable x_i in a pseudotree, its **directly related variables** are:*

$$DRV(x_i) = \{x_i\} \cup \{P(x_i)\} \cup PP(x_i) \tag{4.9}$$

Thus, the directed related variables of variable x_4 in the pseudotree of figure 4.5(b) are defined as $\{x_2, x_3, x_4\}$, where x_2 stands for its pseudoparent, x_3 for its parent and x_4 for its own variable.

On the other hand, the inherited related variables of a node include the variables that its children must send up the tree after eliminating their own variables. Formally:

Definition 12. *Given a variable x_i in a pseudotree, its **inherited related variables** are:*

$$IRV(x_i) = \bigcup_{x_j \in Ch(x_i)} \mathcal{C}_j \backslash \{x_j\} \tag{4.10}$$

Observe that the only variable that a node can remove from a clique's child is its child variable. Notice also that the definition of inherited related variables leads to a recursive definition of cliques and that the set of inherited related variables is empty for leaf nodes.

Thus, the set of inherited related variables of the leaf variable x_4 in the pseudotree of figure 4.5(b) is empty. Then we can assess its clique, $\mathcal{C}_4$, following equation 4.8, as directly its set of directed related variables $\{x_2, x_3, x_4\}$. Once clique $\mathcal{C}_4$ is assessed we can recursively compute the set of inherited related variables of x_3 composed of variables in $\mathcal{C}_4$ excluding x_4, namely $\{x_2, x_3\}$.

Once obtained cliques' variables, we can assess the potentials and separators completing the definition of a junction tree. Thus, finally the mapping γ below allows us to build a junction tree from a pseudotree.

Definition 13 (γ)**.** *Let γ be a function that given a DCOP $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ and a pseudotree $PT = \langle P, PP \rangle$ maps them to a junction tree $\gamma(\Phi, PT) = \langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$, where:*

1. *The set of variables $\mathcal{X}$ is the same as in PT.*
2. *The set of cliques $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|X|}\}$ contains one clique per variable in PT. The clique $\mathcal{C}_i$ contains all the variables directly or inherited related to variable x_i as defined by expression 4.8.*
3. *The set of potentials Ψ contains one potential associated to each clique. Each clique potential ψ_i is the combination of: (i) a unary relation r_i that involves the clique decision variable x_i; and (ii) the binary relations that link x_i with its parent or one of its pseudoparents. Formally:*

$$\psi_i = r_i \otimes \left[\bigotimes_{x_j \in \{P(x_i)\} \cup PP(x_i)} r_{ij} \right] \tag{4.11}$$

4. *The set of separators $\mathcal{S}$ contains one separator s_{ij} per pair of cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ such that x_j is parent of x_i in the PT. By definition of junction tree, each separator s_{ij} contains the intersection of its cliques ($s_{ij} = \mathcal{C}_i \cap \mathcal{C}_j$).*

Figure 4.5(b) shows a pseudotree PT over the DCOP of figure 4.5(a) while figure 4.5(c) shows the junction tree $\gamma(\Phi, PT)$. Observe that mapping γ creates four cliques, one per each variable in the PT and that clique's potentials are assessed following equation 4.11. Thus, the potential of $\mathcal{C}_4$ is composed of the combination of the relation between x_4 and its parent x_3, namely r_{34}, and its pseudoparent x_2, namely r_{24} (no unary relation in this case). Cliques are build recursively by means of equation 4.8. Say now that we have already generated clique $\mathcal{C}_4$, corresponding to variable x_4, and we intend to generate clique $\mathcal{C}_4$, corresponding to variable x_3. Following equation 4.9, the set of DRV of x_3 is $\{x_2, x_3\}$ and, following equation 4.10, its set of IRV is composed of $\mathcal{C}_4$, excluding x_4. Hence $\mathcal{C}_3 = \{x_2, x_3\}$.

Computing mapping γ with the DJTG algorithm

Here we detail how the DJTG algorithm introduced in section 4.3.3 allows agents to distributedly compute mapping $\gamma(\Phi, PT)$. Recall that the DJTG algorithm receives as an input a set of relations distributed among agents and an spanning tree over them. Hence, given a DCOP Φ and a pseudotree PT, agents can compute mapping $\gamma(\Phi, PT)$ by executing the DJTG algorithm setting the input $\langle \mathcal{A}, \mathcal{X}, \mathcal{R}, \kappa, ST \rangle$ as:

- the set of agents $\mathcal{A}$ and variables $\mathcal{X}$ are set as in Φ;
- the set of relations is set as $R = \{\psi_1, \ldots, \psi_n\}$ where ψ_i is defined as in equation 4.11;

- κ maps each agent a_i to ψ_i ($\kappa(a_i) = \psi_i$);
- the spanning tree ST links each pair of relations such that x_j is parent of x_i in the PT.

Hence, executing the DJTG algorithm based on the pseudotree in figure 4.5(b) over the DCOP of figure 4.5(a), results on the DJT shown in figure 4.5(c). By doing so, the DJTG algorithm not only allows to compute mapping γ in a distributed way but also to take advantage of any heuristic defined to generate good pseudotrees in order to generate equivalent good junction trees.

Proving equivalence

The previously introduced mapping (γ) builds a junction tree from each pseudotree. In the remaining of the section we prove that running DPOP over that pseudotree is equivalent to running Action-GDL over the junction tree resulting from applying γ to the pseudotree. First we state (lemma 1) that both the computation performed and the messages exchanged during the utility propagation phase are the same. After that, we state (lemma 2) that the messages exchanged during the value propagation phase are also the same. Finally we combine these two lemmas to prove our main result (theorem 1). For the sake of clarity, in what follows we provide an sketch of the proof for lemmas 1 and 2. Fully detailed proofs are provided in appendix A.

Lemma 1. *Given a DCOP Φ and a pseudotree PT, the computation performed and the messages exchanged during the utility phase of $DPOP(\Phi, PT)$ and Action-GDL($\gamma(\Phi, PT)$) are the same.*

Sketch of the proof.

Proof. We prove the lemma by induction on the depth of the agent in the PT. Both in the base and induction cases, we can prove that: (i) the set of variables handled by agents in both algorithms are the same; and (ii) the domain of the utility messages send by agents in DPOP after eliminating its corresponding variable coincides with separators in Action-GDL. By induction the utility messages received by each agent in both algorithms are the same. This fact along with (i) and (ii) forces that the computation performed and messages exchanged during this phase by each agent must be the same. □

Lemma 2. *Given a DCOP Φ and a pseudotree PT the value assigned by each agent to its variable and the messages exchanged during the value propagation phase of $DPOP(\Phi, PT)$ and Action-GDL($\gamma(\Phi, PT)$) are the same.*

Sketch of the proof.

Proof. We prove the lemma by induction on the depth of the PT. The base case is trivial since there is only one variable in the PT and both algorithms compute the same value for it. In the induction case we can split our PT into the root and a set of PTs of smaller depth. Then: (i) it is easy to see that the root agent acts equivalently in DPOP and in Action-GDL; and (ii) we can apply the induction hypothesis to the PTs of smaller depth. Our result comes from (i) and (ii). □

Lemmas 1 and 2 combined prove the main result of this section:

Theorem 1. *Given a DCOP Φ and a pseudotree PT, the execution of DPOP(Φ,PT) is equivalent to Action-GDL($\gamma(\Phi, PT)$).*

Proof.

Proof. Since both algorithms are only composed of an utility phase and a value propagation phase, the result follows directly from lemmas 1 and 2. □

As discussed in section 4.4.1, computing mapping γ can be done efficiently and distributedly by means of the DJTG algorithm. Then, theorem 1 proves that Action-GDL can be at least as efficient as DPOP in any DCOP (by mimicking its behavior).

4.4.2 Action-GDL generalizes DCPOP

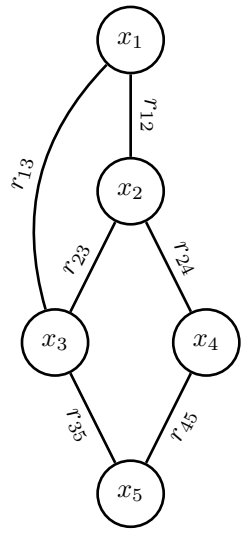
In this section we prove that Action-GDL can be at least as efficient as DCPOP in any DCOP by producing equivalent executions. First, we overview the DCPOP algorithm in section 4.4.2. Next we prove that Action-GDL can produce DCPOP-equivalent executions by: (1) providing a mapping from cross-edged trees (input of DCPOP) to junction trees (section 4.4.2); and (2) proving that given any cross-edged tree, the execution of DCPOP over this cross-edged tree is equivalent to the execution of Action-GDL over the junction tree produced by our mapping (section 4.4.2).

Overviewing DCPOP

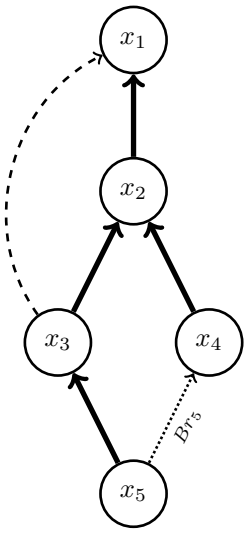
DCPOP (Atlas and Decker, 2007) is a generalization of DPOP based on an extension of pseudotrees, namely cross-edged trees. A cross-edged tree (CT) is a pseudotree with the addition of cross-edges (dotted line). Figure 4.6(b) shows a cross-edged tree for the constraint graph in figure 4.6(a). A cross-edge is an edge from node x_i to node x_j that is above x_i but not in the path from x_i to the root. Thus, besides the parent and pseudoparent relationships, a cross-edged tree adds a new type of relationship: branch-parent/branch-children for variables connected through a cross-edge (for instance, from x_3 to x_4 in figure 4.6(b)). Therefore, we can represent a cross-edged tree as a tuple $\langle P, PP, BP\rangle$, where P, PP, and BP are functions that map each variable to its parent, pseudoparents and branchparents respectively. We obtain function BCh, which returns a variable's branch-children, as $BCh(x_i) = \{x_j \in \mathcal{A} | BP(x_j) = x_i\}$.

Thus, when running DCPOP, agents start with a pre-processing phase to generate a cross-edged tree by running a distributed Best-First Search (BFS) algorithm guided by some heuristic. Then, likewise DPOP, DCPOP has two main phases: to have agents exchange *utilities*, and to have agents propagate *values*.

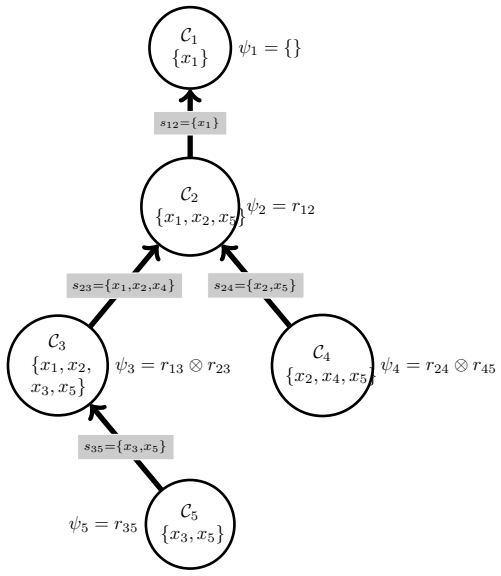
Algorithm 3 shows the phases of DCPOP once a cross-edged tree is generated in terms of the operators introduced in section 4.1. Notice that the algorithm splits the original utility propagation phase in (Atlas and Decker, 2007) into two phases: to propagate branch information and to propagate utility information. This encoding aims at easing the comparison with both DPOP and Action-GDL.



(a) DCOP constraint graph.



(b) Cross-edge tree.



(c) Junction tree $JT = \gamma_{CT}(\Phi, CT)$.

Figure 4.6: Example of constraint graph, cross-edged pseudotree and equivalent junction tree.

In DCPOP the initial knowledge of an agent x_i ($\mathcal{K}_i^0$), besides being composed of some unary relation involving x_i and binary relations linking x_i with one of its parent/pseudoparent variables, also contains binary relations linking x_i with its branch-

Algorithm 3 DCPOP($\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$,$\langle P, PP, BP \rangle$)

Agent $x_i \in \mathcal{X}$ receives $\langle P(x_i), PP(x_i), BP(x_i), \mathcal{K}_i^0 \rangle$ where $\mathcal{K}_i^0 = r_i \otimes \bigotimes_{x_k \in \{P(x_i)\} \cup PP(x_i)} r_{ik} \otimes \bigotimes_{x_k \in BCh(x_i)} r_{ik}$ and runs:

1: **Phase I: Branch information propagation**
2: $Br_i = \langle i, |BP(x_i)| + 1, 1 \rangle$; /*Create branch information for own variable*/
3: Send Br_i to all $BP(x_i)$
4: **for all** $x_k \in BCh(x_i)$ **do**
5: Wait for branch information Br_k from x_k
6: $\langle Br_i, MV \rangle = mergeBranches(Br_i, Br_k)$
7: **end for**
8: **Phase II: UTILITY Propagation**
9: $\mathcal{K}_i = \mathcal{K}_i^0$;
10: **for all** $x_j \in Ch(x_i)$ **do**
11: Wait for utility message $\langle \mu_{ji}, Br_j \rangle$ from x_j
12: $\mathcal{K}_i = \mathcal{K}_i \otimes \mu_{ji}$;
13: $\langle Br_i, MV \rangle = mergeBranches(Br_i, Br_j)$
14: **end for**
15: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
16: Send $\langle \mu_{ip} = \bigoplus_{\setminus MV} \mathcal{K}_i, Br_i \rangle$ to x_p
17: **end if**
18: **Phase III: VALUE propagation**
19: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
20: Wait for a value message σ_{pi} from x_p
21: $\mathcal{K}_i = \bigtriangledown_{\sigma_{pi}} \mathcal{K}_i$; /*Slice $\mathcal{K}_i$ with the value message*/
22: **end if**
23: $x^*_{MV} = arg \max_{d \in \mathcal{D}_{MV}} \mathcal{K}_i(d)$; /* Assess best value for merged variables */
24: $x^* = x^*_{MV} \cup \sigma_{pi}$; /* Put together the assessed values and the message received. */
25: **for all** $x_j \in Ch(x_i)$ **do**
26: Send $\sigma_{ij} = x^*_{Scope(\mu_{ji})}$ to x_j /* Send to x_j the variables he is interested in */
27: **end for**
28: **return** x^*;

children variables. Thus, in figure 4.6(b) the knowledge of agent x_4 is initially composed of relations r_{24} (shared with its parent) and r_{45} (shared with branch-child).

The main operational difference between DPOP and DCPOP has to do with the mechanics that DCPOP incorporates to deal with cross edges during utility propagation. That is because, in DCPOP, a branch-child variable x_i is not eliminated at its node, instead it is eliminated in some node up the tree, at the so-called merge point of x_i. Thus, in DCPOP, each branch-child x_i starts by sending branch information to its branch-parents to calculate the merge point of x_i (lines 2-3). The branch information for a variable x_i contains its identifier, the number of branches of x_i and the number

of merged branches (initially set to 1). After that, each x_i receives and merges branch information from its branch-children (lines 4-7). Next, during the utility propagation phase (lines 8-17), each x_i receives utility messages from all its children variables and combines them with its local knowledge in the very same way as in DPOP. However, in DCPOP, these messages also contain branch information of branch-children variables. Therefore, x_i merges all branches with the same originator by adding up the number of merged branches and assesses the set of variables for which it is merge point (MV), namely variables for which the number of merged branches equals the total number of branches (line 13). At the end of this phase, x_i exchanges a message with its parent x_p (line 16) that contains a utility message that summarizes its current knowledge after filtering out MV and the merged branch information.

Regarding the second message-passing phase, the value propagation phase (lines 18-27), notice that each agent's behaviour is similar as in DPOP with the difference that instead of assessing its own variable, each node x_i assesses all variables in MV.

Mapping cross-edged trees into junction trees

Before proving the equivalence of Action-GDL and DCPOP, in this section we define a mapping that builds a junction tree from a cross-edged tree. First of all, we offer the intuitions behind our mapping from a cross-edged tree to a junction tree. In general, we propose to map each cross-edged tree to a junction tree with as many cliques as nodes in the cross-edged tree. For each node in a cross-edged tree CT, its clique in the corresponding junction tree must contain: (1) the node's variable; (2) the variables expected by the node's parents/pseudoparents up the CT; (3) the variables that the node's children need to forward up the CT; (4) the variables that the node's branch-children need to forward up the CT.

Therefore, notice that the mapping is very similar to mapping γ described in section 4.4.1. The difference lies in the addition of point (4) above involving variables of branch-children and on the set of variables in point (3), which is extended.

Analogously to the approach followed in section 4.4.1, given a node x_i in a cross-edged tree, we can readily define the variables of its clique by wrapping up directly and inherited related variables (see equation 4.8). However, we must extend both sets.

Firstly, the set of directly related variables in equation 4.9 is extended to include the variables that the node's branch-children need to forward the tree. Formally:

Definition 14. *Given a variable x_i in a cross-edged tree CT, its **directly related variables** are:*

$$DRV(x_i) = \{x_i\} \cup \{P(x_i)\} \cup PP(x_i) \cup BCh(x_i) \tag{4.12}$$

Following equation above, the directed related variables of x_4 in the cross-edged tree of figure 4.6(b) are $\{x_2, x_4, x_5\}$ where x_2 stands for its parent, x_4 for its variable and x_5 for its branchchild.
On the other hand, the inherited related variables of a node include the variables that each child must send up the tree after eliminating: (i) those that have already been merged (either by the child or below); and (ii) the child's own variable if it has not branch-parents. Formally:

Definition 15. *Given a variable x_i in a cross-edged tree CT, its **inherited related variables** are:*

$$IRV(x_i) = \bigcup_{x_j \in Ch(x_i)} \mathcal{C}_j \backslash Removable(x_j) \tag{4.13}$$

where $Removable(x_j) = \{x_k | x_k \in \mathcal{C}_j, x_k \neq x_j, x_k$ and all $x_l \in BP(x_k)$ are descendants of $x_j\} \cup \{x_j | BP(x_j) = \emptyset\}$.

Note that the difference between the definition of IRV for a cross-edged tree (equation 4.13) and a pseudotree (equation 4.10) lies in the set of removable variables (the variables that the node's children don't need to forward up the tree). Thus, the set of removable variables of a clique $\mathcal{C}_i$ in a pseudotree is uniquely composed of variable x_i, while in a cross-edged tree is composed of variables whose merge point is $\mathcal{C}_i$ (which includes x_i in case it has not branch-parents).

Say now that we have already generated clique $\mathcal{C}_5 = \{x_3, x_5\}$ in figure 4.6(b) corresponding to variable x_5. Then, by equation 4.13, the set of inherited related variables of its parent x_3 is composed of $\mathcal{C}_5$, but unlike happens in a pseudotree, we can not exclude x_5 because it has a branchparent (x_4) up to x_3 in the tree. Hence, $IRV(x_3) = \{x_3, x_5\}$ in this case.

Next, we formulate function γ_{CT}, which defines the mapping from cross-edged trees to junction trees by providing definitions for potentials and separators in addition to cliques.

Definition 16 (γ_{CT})**.** *Let γ_{CT} be a function that maps a DCOP $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ and a cross-edged tree $CT = \langle P, PP, BP \rangle$ into a junction tree $\gamma_{CT}(\Phi, CT) = \langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$, where:*

- *the set of variables $\mathcal{X}$ is the same as in CT.*
- *the set of cliques $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|X|}\}$ contains one clique per variable in CT. Clique $\mathcal{C}_i$ contains all the variables directly or inherited related to variable x_i.*
- *the set of potentials Ψ contains one potential per clique. Each clique potential ψ_i is the combination of: (i) a unary relation r_i that involves the clique decision variable x_i; (ii) the binary relations that link x_i with its parent and pseudoparents; and (iii) the binary relations that link x_i with its branch-children. Formally:*

$$\psi_i = r_i \otimes \left[\bigotimes_{x_j \in \{P(x_i)\} \cup PP(x_i)} r_{ij} \right] \otimes \left[\bigotimes_{x_j \in BCh(x_i)} r_{ij} \right] \tag{4.14}$$

- *the set of separators $\mathcal{S}$ contains one separator $s_{ij} = \mathcal{C}_i \cap \mathcal{C}_j$ per pair of cliques $\mathcal{C}_i, \mathcal{C}_j$ such that x_j is parent of x_i in CT.*

Figure 4.6(b) shows a cross-edged tree CT over the DCOP Φ of figure 4.6(a) while figure 4.6(c) shows the junction tree $\gamma_{CT}(\Phi, CT)$. Observe that mapping γ_{CT} creates

one clique per variable in the CT and that cliques' potentials are assessed following equation 4.14. Thus, the potential of $\mathcal{C}_4$ is composed of the combination of the relation with its parent x_2, namely r_{24}, and the relation with its branch-child x_5, namely r_{45}.

The example also illustrates how merge points are naturally captured by their corresponding cliques. Notice that x_2 in the cross-edged tree is the merge point for variable x_5. Say now that we have already generated clique $\mathcal{C}_2$ corresponding to variable x_2 and we intend to generate $\mathcal{C}_1$ for variable x_1. Following equation 4.12, the set of DRV of x_1 is uniquely composed of x_1. Next, we apply equation 4.13 to assess the set of IRV of x_1, which in figure 4.6(c) is composed of $\mathcal{C}_2 = \{x_1, x_2, x_5\}$ excluding the set of removable variables at x_2. The set of removable variables at x_2 includes x_2 itself because it has not branch-parents, and x_5 because both x_5 and its branch-parent x_4 are descendants of x_2. Hence $\mathcal{C}_1 = \{x_1\}$.

In general, if a variable x_i in a cross-edged tree is the merge point for another variable x_j, our mapping guarantees that $\mathcal{C}_i$ eliminates variable x_j. Therefore, because variables' merge points are explicitly represented, the junction tree produced by our mapping saves both the computing and sending of branch information.

Computing mapping γ_{CT} with the DJTG algorithm

Here we detail how the DJTG algorithm introduced in section 4.3.3 allows agents to distributedly compute mapping $\gamma_{CT}(\Phi, CT)$. Recall that the DJTG algorithm receives as an input a set of relations distributed among agents and an spanning tree over them. Hence, given a DCOP Φ and a cross-edge tree CT, agents can compute mapping $\gamma_{CT}(\Phi, CT)$ by executing the DJTG algorithm with the input $\langle \mathcal{A}, \mathcal{X}, \mathcal{R}, \kappa, ST \rangle$ where:

- the set of agents $\mathcal{A}$ and variables $\mathcal{X}$ are set as in Φ;
- the set of relations is set as $\mathcal{R} = \{\psi_1, \ldots, \psi_n\}$ where psi_i is defined as in equation 4.14;
- κ maps each agent a_i to ψ_i ($\kappa(a_i) = \psi_i$);
- the spanning tree ST links each pair of relations such that x_j is parent of x_i in the CT.

Hence, executing the DJTG algorithm based on the cross-edge tree in figure 4.6(b) over the DCOP of figure 4.6(a), results on the distributed junction tree shown in figure 4.6(c). By doing so, the DJTG algorithm not only allows to compute mapping γ_{CT} in a distributed way but also to take advantage of any heuristic defined to generate good cross-edge trees in order to generate equivalent good junction trees.

Proving equivalence

Analogously to the equivalence analysis involving Action-GDL and DPOP, in this section we analyse the relationship between DCPOP and Action-GDL. We argue that running DCPOP over a cross-edged tree is equivalent to running Action-GDL over its (as produced by mapping γ_{CT}) junction tree whenever the computing and sending of

branch information is disregarded. As argued above, such information is not required because in a junction tree the merging points of variables are explicitly represented.

Under this assumption, we obtain analogous equivalence results to those obtained for DPOP in section 4.4.1.

Lemma 3. *Given a DCOP Φ and a cross-edged tree CT, the computation performed and the messages exchanged during the utility phase of DCPOP(Φ, CT) and Action-GDL($\gamma_{CT}(\Phi, CT)$) are the same disregarding the computing and sending of branch information.*

Lemma 4. *Given a DCOP Φ and a cross-edged tree CT the value assigned by each agent to its variable and the messages exchanged during the value propagation phase of DCPOP(Φ, CT) and Action-GDL($\gamma_{CT}(\Phi, CT)$) are the same.*

We can build the proof for lemmas 3 and 4 following the same approach as used on proving lemmas 1 and 2 in section 4.4.1. Here we only comment on the intuitions behind these proofs.

Regarding lemma 3, if there are no cross-edges, DCPOP behaves like DPOP. If there are cross-edges, branch-children variables are eliminated on their merge points, namely on the lowest variables in the cross-edged tree that are between them and the root and between their branch-parents and the root. As argued above, Action-GDL does not require branch information because merge points are explicitly represented in cliques and separators of the junction tree generated by mapping γ_{CT}. Thus, in the junction tree $\gamma_{CT}(\Phi, CT)$, the set of variables whose merge point is x_i are the variables in $\mathcal{C}_i$ that are not in the separator with its parent s_{ip}. Hence, if we focus on comparing the computing and sending of utility information as well as on the computing of local knowledge, we observe that the utility phase of DCPOP(Φ, CT) and Action-GDL($\gamma_{CT}(\Phi, CT)$) are the same.

Regarding lemma 4, since variables assessed at some node x_i are the set of variables for which x_i is a merge point which lemma 3 states that are correctly captured by our mapping γ_{CT}, it is rather straightforward that lemma 4 holds.

The combination of lemmas 3 and 4 leads to the following equivalence theorem:

Theorem 2. *Given a DCOP Φ and a cross-edged tree CT, the execution of DCPOP(Φ, CT) is equivalent to Action-GDL($\gamma_{CT}(\Phi, CT)$) disregarding the computing and sending of branch information.*

Likewise mapping γ, since in section 4.4.2 we have shown that computing of mapping γ_{CT} can be done efficiently and distributedly by means of the DJTG algorithm, we can consider the overhead of computing the mapping negligible with respect to the time of solving the DCOP. Therefore, theorem 2 proves that Action-GDL can be at least as efficient as DCPOP in any DCOP.

4.5 Characterizing Action-GDL usefulness

From theorems 1 and 2 we conclude that we can obtain no benefit from using DPOP and DCPOP over Action-GDL. Now the question is: can Action-GDL improve

DPOP/DCPOP in terms of (i) the computational/communication needs required from the agents or (ii) the degree of parallelism when solving a DCOP?

Next, in section 4.5.1 we provide some theoretical results that help answer the first question with respect to DPOP. Moreover, from these theoretical results we obtain some insights regarding how to exploit the space of junction trees effectively. Thus, in section 4.5.2 we propose a postprocessing of junction trees to improve the computation, communication and degree of parallelism of a junction tree. In section 4.6 we empirically show that such postprocessing helps Action-GDL significantly outperform DCPOP over the best cross-edged tree/pseudotree generated out of multiple heuristics.

4.5.1 Theoretical improvements with respect to DPOP

In this section we provide theoretical results showing in which cases Action-GDL can outperform DPOP in terms of communication and computation.

Action-GDL provides significant savings in computation over DPOP when pseudotrees are generated by edge-traversal heuristics

In (Atlas and Decker, 2007) Atlas and Decker show by means of an example that there exists DCOP instances for which a cross-edged tree significantly outperforms all possible pseudotrees based on edge-traversal heuristics. Because, by theorem 2, Action-GDL execution is equivalent to DCPOP execution when it runs over a γ_{CT} mapping JT, Action-GDL can also benefit from this result with respect to DPOP.

Action-GDL provides no significant savings in computation for unrestricted pseudotrees

In this subsection we prove that for any DCOP, given a junction tree, we can always construct a pseudotree so that the amount of computation for DPOP is of the same order of magnitude than that of Action-GDL.

Lemma 5. *Given a DCOP Φ and a junction tree, algorithm 4 computes a pseudotree such that the computational requirements of DPOP are of the same order of magnitude than those of Action-GDL (the size of the largest table to be maximized is the same)*

Proof. First we have to ensure that the tree constructed by algorithm 4 is a pseudotree, that is, we have to check that adjacent nodes in the constraint graph fall in the same branch of the tree. Let x_i and x_j be two adjacent nodes in the constraint graph. By virtue of the covering property, there should be a node of the JT that contains both x_i and x_j. If this is the highest node where both x_i and x_j appear then they will be placed in a chain and hence they will be in the same branch of the tree (lines 9-10). Otherwise, assume without loss of generality that x_i appears in a node in a different branch. By the running intersection property, x_i must appear also in the root of the subtree containing these two nodes. By construction, the branch for x_j will never be inserted into PT before the appearance of x_i. Hence, both x_i and x_j appear in the same branch (the one that has x_i as root) (line 13-15). Then, our main claim can be proven by induction on the number of variables of JT. If there is a single variable both algorithms are equivalent

Algorithm 4 JT2PT(JT)

1: $\mathcal{C}_k =$ Find the largest clique in JT
2: $JT' = JT$ rooted at the agent responsible for $\mathcal{C}_k$
3: $T = GenerateSpanningTree(JT', \emptyset)$;
4: $PT =$ Construct the PT corresponding to T;
5: return PT
6:
7: **function** GenerateSpanningTree(JT,V)
8: $\mathcal{C}_k = getRoot(JT)$ /*Let $\mathcal{C}_k$ be the root of JT */
9: $Scope(\mathcal{C}_k) \setminus V = \{x^1, \ldots, x^m\}$ /*Establish an order among the variables in $\mathcal{C}_k$ not in V*/
10: $T = \{(x^i, x^{i+1}) | 1 \leq i < m\}$ /*Include into T a chain linking variables in $\mathcal{C}_k$ not in V */
11: **for all** $JT_i \in Subtree(JT, \mathcal{C}_k)$/*For each subtree of JT, one for each child of $\mathcal{C}_k$*/ **do**
12: $T_i = GenerateSpanningTree(JT_i, V \cup Scope(\mathcal{C}_k))$
13: **if** $Scope(T) \cap Scope(JT_i) \neq \emptyset$ **then**
14: $j = \max\{k | 1 \leq k < m \text{ and } x^k \in JT_i\}$ /*Find JT_i variable with lowest position in T*/
15: **else**
16: $j = m$
17: **end if**
18: $T = T \cup T_i \cup \{x^j, Root(T_i)\}$/* Include T_i into T by linking its root as a child of x^j */
19: **end for**
20: **return** T;

and hence our result holds. If JT has more than one variable then the computational requirements to run DPOP in the subset of PT composed by the variables of the largest clique (appearing as a chain hanging from the root of the PT) are of $\mathcal{O}(d^m)$ (where m is the size of the clique and d is the highest cardinality of any variable in the clique). By induction hypothesis this is also the case in each of the subpseudotrees hanging from variables in the largest clique. It is easy to see that the size of the largest table to be maximized for Action-GDL is also $\mathcal{O}(d^m)$ □

This result does not mean that the processing of Action-GDL and DPOP will be the same but ensures that the improvement that we can expect from Action-GDL cannot be very large. However, there is no mention on the amount of messages exchanged. In fact, our next result proves that Action-GDL can effectively improve on that.

Action-GDL can severely reduce communication complexity

In this subsection we show that there are DCOPs for which Action-GDL severely reduces the amount of communication with respect to DPOP. Concretely, we prove that

when the DCOP is composed of a single utility relation involving all variables[5], the amount of communication required grows linearly with the number of variables for Action-GDL using the best junction tree and grows exponentially for DPOP with any pseudotree.

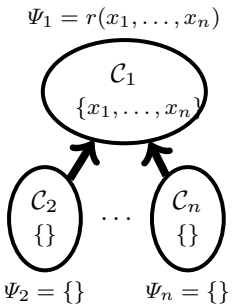


Figure 4.7: Best junction tree.

Lemma 6. *Given a DCOP* $\Phi = \langle \mathcal{X} = \{x_1, \ldots, x_n\}, \mathcal{D}, \mathcal{R} = \{r\} \rangle$ *such that* $Scope(r) = \mathcal{X}$*, the amount of communication required to run Action-GDL using the junction tree depicted in figure 4.7 grows linearly in the number of variables.*

Proof. In the utility propagation phase $x_2, \ldots, x_n$ send empty messages to x_1. Then x_1 computes the overall solution and distributes the decisions to $x_2, \ldots, x_n$ in the value propagation phase, exchanging $n-1$ messages, the largest of them being of size $\log d$, where $d = \max_i |D_i|$. Hence the amount of communication is $\mathcal{O}(n \log d)$. □ □

Lemma 7. *Given a DCOP* $\Phi = \langle \mathcal{X} = \{x_1, \ldots, x_n\}, \mathcal{D}, \mathcal{R} = \{r\} \rangle$ *such that* $Scope(r) = \mathcal{X}$*, the amount of communication required for DPOP independently of the pseudotree grows exponentially in the number of variables.*

Proof. First note that the only possible structure for a pseudotree is a chain, because otherwise adjacent vertices in the graph will appear in different branches (since all vertices are adjacent). There are as many pseudotrees as variable orderings. Assume without loss of generality that the ordering places x_1 in the root, then x_2 as its child and so on until x_n as a single leave. DPOP places the relation r in x_n. The execution starts maximizing r with respect to x_n. The computed relation is sent to x_{n-1} which maximizes it with respect to x_{n-1} and the process continues that way until it reaches x_1. Then the best value for x_1 is computed and sent to x_2 where the best value for x_2 is computed and sent to x_3 together with the optimal value for x_1 and the process continues that way until it reaches x_n. The algorithm exchanges $n-1$ utility messages, the largest of them of size d^{n-1} and $n-1$ value messages, the largest of them of size $\sum_{i=1}^{n-1} \log |D_i|$. Hence, the overall amount of communication is $\mathcal{O}(d^n)$. □ □

[5]Note that the relation containing all variables does not result from any partial centralization of the algorithm, instead it is formulated like this in the original DCOP.

Lemmas 6 and 7 prove that Action-GDL can severely improve DPOP communication complexity. Furthermore, it suggests that for more complex graphs, the improvement could be related to the treewidth of the constraint graph.

4.5.2 Postprocessing junction trees

Taking inspiration on lemmas 6 and 7, we propose to postprocess the junction tree constructed by the γ_{CT} mapping to reduce the amount of computation, the sizes of messages and the degree of parallelism. Firstly, in order to reduce the amount of computation and the size of messages we propose to exchange two connected cliques in the junction tree, namely $\mathcal{C}_i$ and its parent $\mathcal{C}_p$, following the transformation depicted in figure 4.8(a), whenever the set of variables in $\mathcal{C}_p$ is a subset of $\mathcal{C}_i$. Formally:

$$\mathcal{C}_p \subseteq \mathcal{C}_i \tag{4.15}$$

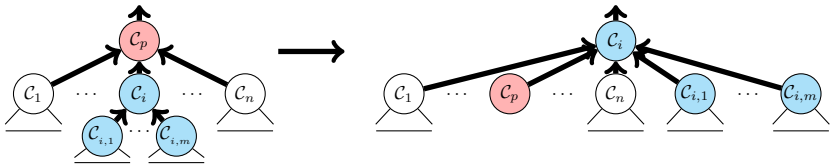
After swapping parent for child, the child takes its parent's children but keeping also its own children as depicted on the right hand side of figure 4.8(a). The intuition behind the transformation is straightforward. Since the structure in figure 4.7 is the best one for processing a clique with Action-GDL, whenever there is a clique whose variables are included into one of its children, we can think of swapping parent for child. Figure 6 (a)(b) depict the two transformations carried out by our postprocessing over the junction tree in figure 4.5(c). The first transformation only swaps $\mathcal{C}_4$ and $\mathcal{C}_3$ without involving any deeper change. The second transformation entails a more profound rearrangement because in order to swap $\mathcal{C}_4$ for $\mathcal{C}_2$, $\mathcal{C}_4$ must keep $\mathcal{C}_3$ as a child.

Notice that if we start from a valid junction tree, the resulting junction tree after this transformation still satisfies the running intersection property (RIP) without increasing any clique. Furthermore, it is likely that cliques can be reduced after the swap by deleting some variables not longer necessary to ensure the RIP. Concretely, after the transformation clique C_p can be restricted to deal only with variables in the scope of its potential, that is $Scope(\psi_p)$, thus reducing the amount of computation and size of messages for C_p. That is because after a swap, C_p is always a leaf node so it will not have to enlarge its clique to carry variables to satisfy the RIP. Thus, in the example of figure 4.8(c), as a consequence of the change of position, $\mathcal{C}_2$ can delete x_2 from its set of variables.

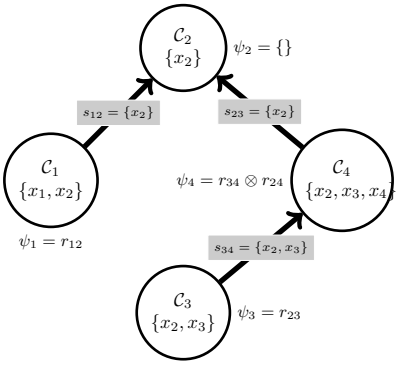
To summarise, our postprocessing performs a postorder tree traversal of the JT computed by γ_{CT}, applying the transformation depicted in figure 4.8(a) whenever the condition in equation 4.15 holds. Hence, its distributed implementation is direct (Santoro, 2006).

Secondly, after the postorder traversal, we select the root of the junction tree that maximises the degree of parallelism (the maximum amount of sequential computation required by agents when running Action-GDL). This last step is important because although changing the root of a JT does not change the amount of computation nor of messages exchanged, it can modify its degree of parallelism.

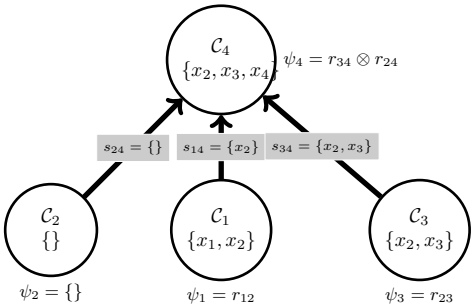
Observe that the resulting junction tree in figure 4.8(c) reduces communication, computation, and improves parallelism with respect to the original junction tree in figure 4.5(c).



(a) Transforming a junction tree by clique swap.



(b) $\mathcal{C}_4$ swapped for $\mathcal{C}_3$.



(c) $\mathcal{C}_4$ swapped for $\mathcal{C}_2$.

Figure 4.8: (a) Postorder transformation and (b,c) transformations of the junction tree in figure 4.5(c).

Postprocessing complexity

In what follows we assess the complexity of the postprocess methods described above. Firstly, in the postorder tree traversal the information exchanged is $O(n^2)$ (each node that swaps exchanges messages with all its neighbours) and the overall computation is $O(n^2)$ where n is the number of variables in the DCOP. Secondly, to distributedly select the root of the pseudotree, agents can execute a distributed leader election algorithm (Barbosa, 1996) which information exchanged and overall computation is $O(n)$. Therefore we can conclude that: (1) the postprocessing can be computed distributedly, and; (2) the overhead introduced is not significant with respect to the costs of solving the DCOP.

4.6 Empirical evaluation

In this section we aim at providing evidence that using Action-GDL instead of DCPOP (or DPOP) is useful from a practical point of view. In (Atlas and Decker, 2007) Atlas and Decker provide empirical evidence of the significant improvements that DCPOP can obtain when compared to DPOP. Since ActionGDL generalizes DCPOP, it can also benefit from the same improvements with respect to DPOP. Thus, in our experiments,

we directly compare Action-GDL with DCPOP.

4.6.1 Measures of interest

Following the definition of efficiency discussed in chapter 1 for complete DCOP algorithms, we are interested in comparing DCPOP and Action-GDL regarding the amount of communication, computation, and parallelism required in an experimental scenario. Since we have proved that Action-GDL is a generalization of DCPOP, the metrics defined below for Action-GDL can be readily used for DCPOP.

Computation. The amount of computation at node i is assessed as the sum of the product of the domains' cardinality of variables in its clique, $MC_i = \prod_{x_k \in \mathcal{C}_i} |\mathcal{D}_k|$. The total amount of computation is $\sum_{i=1}^{n} MC_i$.

Communication. The size of a utility message μ_{ij} is $\prod_{x_k \in s_{ij}} |\mathcal{D}_k|$. As noted in (Atlas and Decker, 2007), most communications in DCPOP are utility messages. This is also true for Action-GDL. Hence, we have disregarded value messages in our comparison because they only add a small constant factor. As with computation, we assess the overall amount of communication by adding the size of every message.

Parallelism. Since both DCPOP and Action-GDL are distributed algorithms, we are also interested in the degree of parallelism that we can obtain in its processing. Following (Atlas and Decker, 2007), we measure the degree of parallelism using the maximum path cost (MPC) that measures the maximum amount of sequential computation to perform. The maximum path cost for a given junction tree is defined as $MPC = \max_i \sum_{\mathcal{C}_j \in P_i} MC_j$ where P_i is the path from the root of the junction tree to clique $\mathcal{C}_i$.

4.6.2 Experimental design and results

In the experiments we use four heuristics to generate DCPOP cross-edged trees: (1) DFS-MCN (Depth-First Search Maximum Connected Node) heuristic (Petcu, 2007), which generates pseudotrees; and (2) BFS-MCN (Best-First Search Maximum Connected Node), BFS-LCN (BFS Less Connected Node) and BFS-A-B (BFS Ancestors\ Branch-parents\Branch-children rule) heuristics (Atlas and Decker, 2007) that generate cross-edged trees.

For DCPOP we chose the best cross-edged tree produced by these heuristics. These pseudotrees/cross-edged trees are subsequently input to the γ_{CT} mapping to generate junction trees which are further postprocessed as explained in section 4.5.2 to obtain the input for Action-GDL. For Action-GDL we chose the best junction tree produced by this post-processing.

We empirically compare DCPOP with Action-GDL by plotting the average of the percentual improvement of ActionGDL with respect to DCPOP for each metric. We assess the percentual improvement as $P = \frac{(A-D)}{(A+D)} \cdot 200$, where A is the value of the chosen metric for ActionGDL and D stands for the value for DCPOP.

Scenario	Meetings	# Var.	# Dom	Den.	Comp.	Comm.	MPC
A/1	8	23	9	1.9	2.3%	22.6%	5.6%
B/2	10	26	9	1.8	9.2%	134.2%	3.0%
C/3	12	71	9	1.7	2.8%	28.0%	31%
D/4	12	72	9	1.7	2.3%	23.0%	21.4%

Table 4.4: Results for the different scenarios of the meeting scheduling dataset.

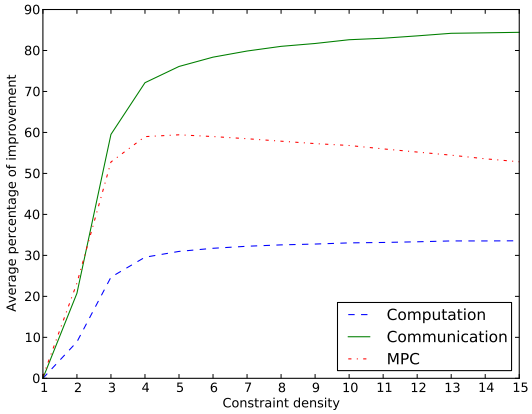
4.6.3 Generic DCOP instances

Our initial tests perform a comparison over randomly generated DCOPs with binary variables. We analyse the differences between DCPOP and Action-GDL as we increase the number of constraints as well as the number of variables. Thus, we characterize each scenario by a number of variables n and a constraint density d. For each scenario, we generate 10.000 random problems. We have explored scenarios with n ranging from 10 to 100 in 10 steps increments and d ranging from 1 to 15 in 1 step increment.

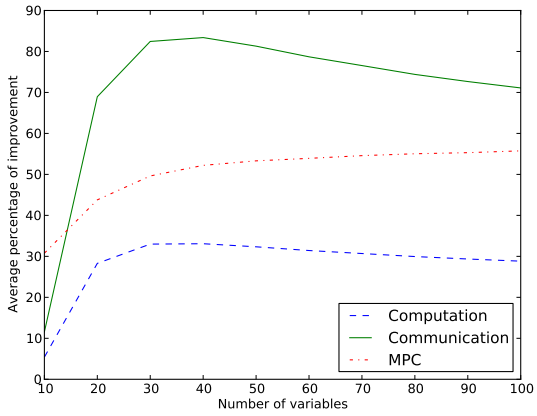
Figure 4.9 summarizes our experimental results. Figure 4.9(a) shows the average of percent improvement among tests as the constraint density increases. We observe that the denser the DCOP, the larger the improvement of Action-GDL regarding communication and computation with respect to DCPOP. Concretely, Action-GDL reduces communication up to around 85%. The amount of computation is not reduced so significantly, though we still obtain average percent improvements of around 30%. We also measured the computation and communication improvement as in Atlas and Decker (2007) in terms of the average of the difference in the number of dimensions. Using these metrics the experiments show improvements up to 10 dimensions. With respect to the improvement on computation and communication, the improvement on the degree of parallelism behaves differently: observe that the MPC reaches the highest value, around 60%, when density is set to 4, and after that it smoothly decreases up to around 50%. That result is explained because in denser DCOPs it is more likely that there is a single clique with a larger number of dimensions than others. Then, this clique conditions the MPC no matter the problem representation. The difference reported for these metrics between ActionGDL and DCPOP is statistically significant within a single value of density (paired Student's t-tests calculate $p < 0.05$) except for density 1.

Moreover, we also show in figure 4.9(b) the average of percent improvement as the number of variables increases. We observe that Action-GDL reaches the higher computation/communication improvement with respect to DCPOP, around 80% and 30% respectively, in medium size DCOPs (with 30-40 variables). As the number of variables increases the average of improvements tend to around 70% and 28% on respectively. In terms of the average of the difference in the number of dimensions, these results implies an improvement of up to 16 dimensions. With regard to the degree of parallelism, we observe that MPC increases with the number of variables up to 55%. We run paired Student's t-tests and the difference between the metrics between ActionGDL and DCPOP is statistically significant within a single value of variables ($p < 0.05$) .

To sum up, the cost of solving random DCOPs is significantly reduced respect to DCPOP when running Action-GDL over the postprocessed junction trees mapped from



(a) Average percentual improvement as constraint density increases



(b) Average percentual improvement as the number of variables increases

Figure 4.9: Action-GDL improvement over DCPOP in computation, communication and MPC

best cross-edged trees. Next we show that such improvement is specially significant for dense problems.

4.6.4 Meeting scheduling dataset

Besides the generic DCOP tests, we also run additional tests on a meeting scheduling dataset, a common problem used by the DCOP community. Concretely, we use the meeting scheduling dataset from (Maheswaran et al., 2004b), publicly available in (Yin, 2008). This dataset is composed of four scenarios (labeled as A/1,B/2,C/3 and D/4), which correspond to four different topologies, with 30 different instances per scenario.

Table 4.6.3 shows the results for the meeting scheduling dataset as well as the characteristics of each scenario (the number of variables/constraints, the cardinality of the variables' domain, etc). All scenarios are composed of sparse problems with a constraint density lower than 2. The results obtained in the meeting scheduling dataset are in line with those obtained for generic DCOPs for similar scale and density and are also statistically significant ($p < 0.05$ for all paired Student's t-tests). Firstly, the MPC is around $3-10\%$ in small scenarios (8-10 variables) and around $20-30\%$ for larger scenarios (70 variables). Thus, as shown in figure 4.9(b) for random instances, the MPC increases with the number of variables. With regard to the improvement on computation, it is less than 10% in all scenarios, with similar values to those shown in figure 4.9(a) when density is set to 2. Finally, the improvements on communication are, in most scenarios, close to those reported in figure 4.9(a) for random instances of density 2, with average percent improvements of around $20-30\%$. However, in scenario $C/3$ we obtain a much higher average percentual improvement. Therefore, although one can characterize the average improvement given the density and the scale of the problem, we observe that the topology of the constraint graph is also an important factor. In particular, our results showed that the communication improvement on some structured topologies is significantly larger than on random ones.

4.7 Conclusions

In this chapter we provided solutions to overcome the limitations of complete dynamic programming DCOP algorithms discussed in chapter 3. Along this line, the main contribution of this chapter was Action-GDL, a novel complete DCOP algorithm that exploits a distributed junction tree representation of the DCOP. Action-GDL was formulated as an extension to the GDL framework to solve DCOPs efficiently, reducing the required communication (the number and size of messages) and computation. In what follows we list the solutions provided by Action-GDL to the open questions listed in section 3.4 regarding the limitations of complete DCOP approaches to exploit more general problem representations.

Firstly, we showed the generality of Action-GDL by proving that it generalises DPOP and DCPOP. With this aim, we provided with two mappings that connect the spaces of problem representations used by these algorithms: (i) a mapping from pseudotrees (used by DPOP) to junction trees; and (ii) a mapping from cross-edge trees (used by DCPOP) to junction trees.

Secondly, we theoretically and empirically characterise the potential benefits from using junction trees with Action-GDL instead of pseudotrees or cross-edged trees. On the one hand, we provide some theoretical results that prove that using junction trees instead of pseudotrees leads to significant benefits in terms of computation and communication. In particular, we observed that Action-GDL: (i) provides significant savings in computation over DPOP when pseudotrees are generated by edge-traversal heuristics; (ii) provides no significant savings in computation for unrestricted pseudotrees; and (iii) can severely reduce communication complexity. On the other hand, we characterise the empirical benefits of Action-GDL with respect to DCPOP. With this aim we propose a novel distributed heuristic to post-process junction trees. Finally, we empirically show that Action-GDL significantly outperforms DCPOP when running over the junction trees that results from post-processing the best cross-edged pseudotrees DCPOP can operate on. Concretely, we observed that our distributed post-processing heuristic allows Action-GDL to outperform DCPOP by: (i) decreasing communication (up to around 85%); (ii) reducing computation (up to around 30%); and (iii) increasing parallelism (up to around 60%).

Thirdly, we argue that several analytical benefits stem from the generality of the GDL framework. In particular, by exploiting this generality, Action-GDL may benefit from: (i) connections with well-known algorithms used in other communities (e.g. Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988)); and (ii) a wealth of theoretical results for GDL over junction trees (Aji and McEliece, 2000). Specifically in the DCOP community, we show how Action-GDL builds a bridge between dynamic programming DCOP algorithms (DPOP and DCPOP) and some incomplete DCOP algorithms also based in GDL, namely Max-Sum and Bounded Max-Sum.

Figure 4.5 shows the resultant DCOP landscape after incorporating the aforementioned contributions of this chapter. Observe that now this landscape includes Action-GDL which subsumes DPOP and DCPOP algorithms by handling distributed junction trees. Moreover, all the dynamic programming complete algorithms are unified under the GDL-framework establishing connections with the GDL-based incomplete algorithms, max-sum and bounded max-sum.

In this chapter we focused on optimal dynamic programming approaches (on the upper left-side of the DCOP landscape of figure 4.5) and dealt with the problem of designing efficient complete DCOP algorithms by means of exploiting more general problem representations. However, as argued in chapter 1, optimal approaches typically do not scale to large systems or apply to domains with very limited resources. Therefore, with the aim of providing solutions for these domains, in the following chapters we focus on the complementary challenge of designing incomplete algorithms with quality guarantees.

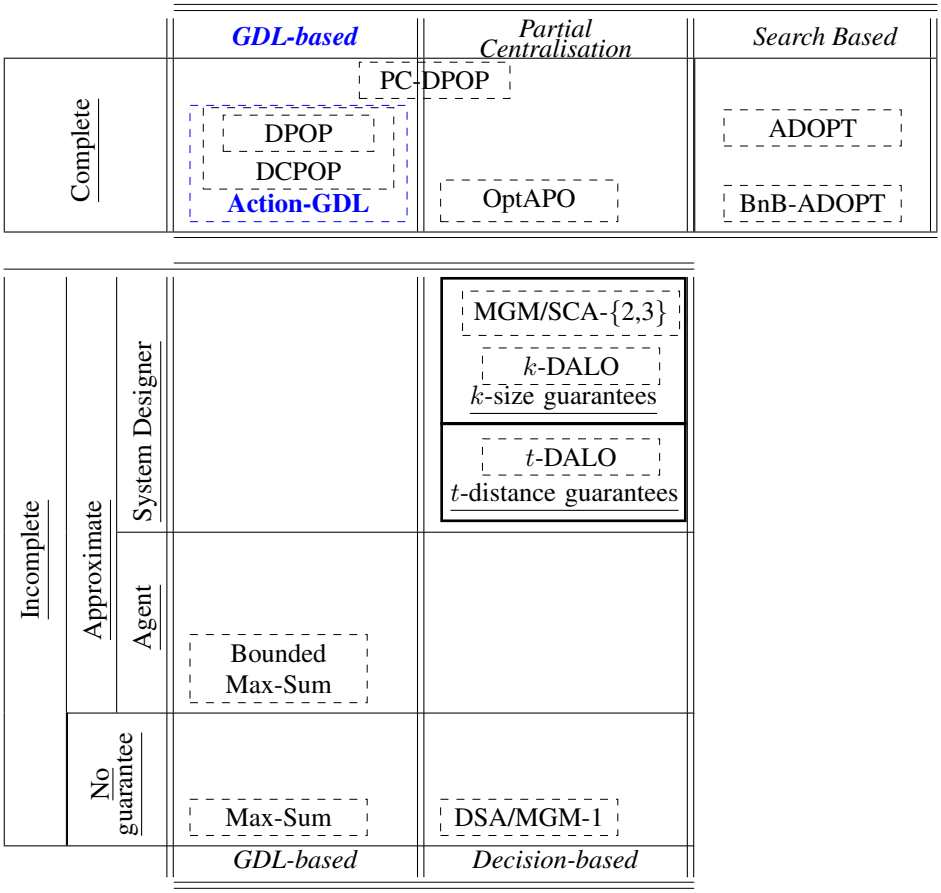


Table 4.5: DCOP algorithms landscape after Action-GDL. Contributions of this chapter are highlighted in bold/blue. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

Chapter 5

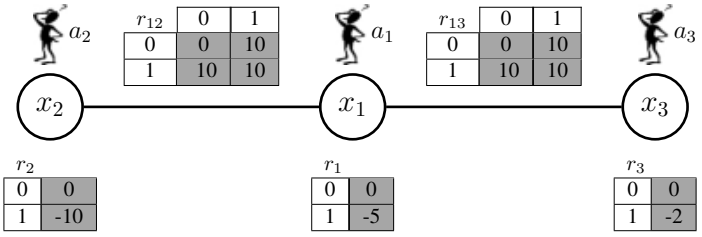
Divide-and-Coordinate

Complete algorithms, such as the Action-GDL algorithm proposed in chapter 4, have the advantage of returning the global optimal solution. However, the cost of this completeness often limits their applicability to actual-world domains. To address domains where scalability and efficiency are of primary importance, this chapter describes a new family of incomplete DCOP algorithms that can return fast bounded solutions.

The main contribution of this chapter is a new family of incomplete DCOP algorithms that uses a novel approach, the *Divide* and *Coordinate* (DaC) approach, to solve DCOPs. Solutions assessed by the DaC approach come with per-instance quality guarantees. As discussed in chapter 1, agents can use to trade-off quality versus cost and/or reduce their uncertainty at run time. The key idea behind DaC is to divide DCOP into subproblems that can be solved independently by each agent with the goal of finding a division in which agent's local solutions agree. With the aim of getting closer to such agreement, DaC agents: (i) coordinate by exchanging information about their local subproblems; and (ii) update their subproblems based on that information creating a new division of the DCOP. Hence, the DaC approach leads to different DaC algorithms depending on: (i) the information exchanged; and (ii) how agents update their subproblems based on such information.

This chapter formulates two DaC algorithms: DaCSA and EU-DaC. In DaCSA agents coordinate by exchanging the most basic local information that allow them to identify the conflicts on assignments: their local solutions. To improve DaCSA performance, we propose EU-DaC where agents coordinate by exchanging the utilities of their local assignments instead of only the solutions. We benchmark these DaC algorithms with other state-of-the-art DCOP algorithms to compare solution qualities and tightness of DaC quality guarantees.

This chapter is structured as follows. Section 5.1 formalises the *Divide-and-Coordinate* approach. Next, section 5.2 introduces a generic DaC algorithm that founds the different realisations of the DaC approach. Section 5.3 describes the formal foundations of DaCSA along with the algorithmic details of its particular realisation of the generic DaC algorithm. Section 5.4 introduces EU-DaC along similar lines. Finally, section 6.3 details an empirical evaluation of the DaC algorithms, and section 5.6 summarises this chapter contributions and draws some conclusions.



Optimal solution: $\mathbf{x}^* = \{\mathbf{x_1 = 1}, \mathbf{x_2 = 0}, \mathbf{x_3 = 0}\}$, global reward: $\mathbf{R(x^*) = 15}$

Figure 5.1: Example of a DCOP.

5.1 Divide-and-Coordinate framework

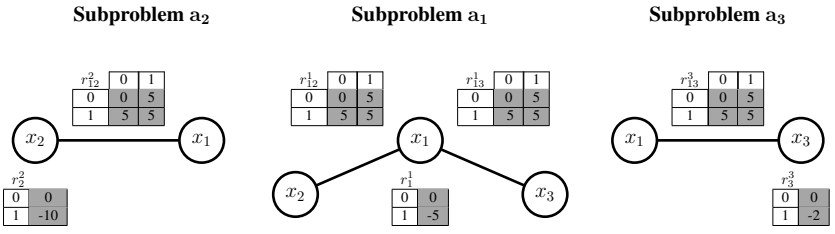
With the aim of providing a bounded, approximate algorithm for DCOPs, in this section we define the *divide-and-coordinate* (DaC) approach. First, in section 5.1.1 we describe the operation of the Divide-and-Coordinate approach through examples and general intuitions. Then, in section 5.1.2 we detail the formal foundations and proofs for that approach.

5.1.1 Divide-and-Coordinate: the approach

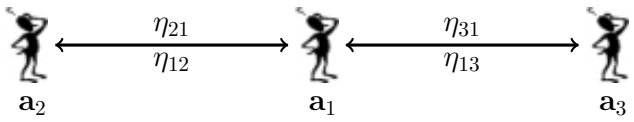
DaC agents aim to solve a DCOP by exploiting the concept of agreement. Figure 5.1 shows a binary DCOP of three variables in which each agent chooses values for its variables from $\{0, 1\}$. Each relation shows its rewards in a table. Thus, agent a_3 has a reward of -2 to set its variable x_3 to 1, and each pair of agents has a reward of 10 to set to 1 at least one of their variables. Following the DCOP model introduced in Chapter 2, relation r_{12} is known by agent a_1, which controls variable x_1, and agent a_2, which controls variable x_2. Likewise, relation r_1 is only known by agent a_1, which controls variable x_1. It is easy to see that the optimal solution of DCOP in figure 5.1 (x^*) is obtained by setting x_1 to 1 and the rest of variables to 0, with a global reward $R(x^*)$ of 15.

The key idea behind the DaC approach is the following: since solving a DCOP is NP-Hard, we can think of *dividing* this intractable problem into simpler subproblems that can be individually solved by each agent. Figure 5.2(a) shows the subproblems created by agents when dividing the DCOP in figure 5.1 as well as the local solutions obtained when individually solving these subproblems. Each agent uses its local relations to create its subproblem. For instance, the local problem of agent a_1 is composed of its local relation r_1 over its variable x_1 and all binary relations shared with its neighbours (r_{12}, r_{23}). For instance, in figure 5.2(a), a_1 and a_2 take one half each of relation r_{12}. In this way, agents never double-count rewards.

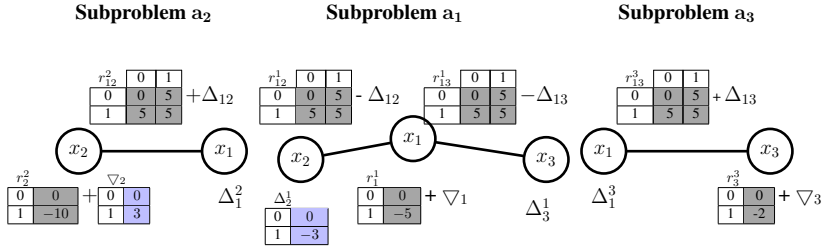
When solving individual subproblems, agents may assign different values to their shared variables, thus causing conflicts between assignments. For instance, as shown



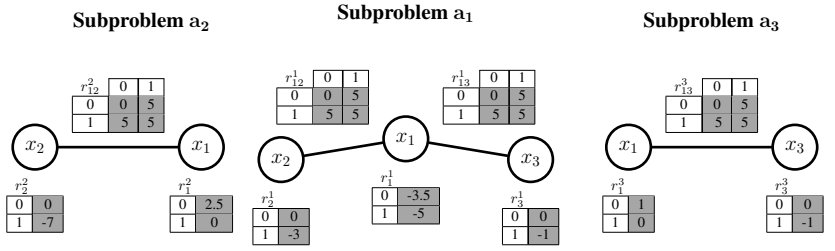
(a) Initial division



(b) DaC coordinate stage.



(c) New subproblems (division) after the divide stage.



(d) Division with agreement.

Figure 5.2: Trace of DaC over the DCOP in figure 5.1.

by the local assignments in figure 5.2(a), agent a_1 conflicts with a_2 and a_3 on the value of x_1 (compare $x^{*,1}$ with $x^{*,2}$ and $x^{*,3}$). Thereafter, each agent proceeds to coordinate, during the so-called *coordinate* stage, by exchanging information about the conflicts on the assignments of the shared variables with its neighbours. For instance, in figure 5.2(a), a_1 will exchange information about its conflict over x_1 with a_2 and a_3. Agents subsequently employ information on disagreements to jointly update their subproblems, during the *divide* stage, to move closer and closer to an agreement. Hence, agent a_1 will use the information exchanged about its conflicts over x_1 with a_2 and a_3 to update its subproblem. At any point in time, the DaC framework requires subproblems to be a *division* of the original DCOP: combining the relations splitted in different subproblems produces the original relations. Therefore, the DCOP rewards are never lost or double counted among agents. As we formally prove in the next section, if all agents reach an agreement on a joint solution when optimizing their local subproblems, then this solution stands for the optimal DCOP solution. For instance, figure 5.2(d) shows an example of a division of the DCOP in figure 5.1 into three subproblems whose individual solutions agree on the assignment $x_1 = 1, x_2 = 0, x_3 = 0$, which is the optimal assignment of the DCOP in figure 5.1. DaC agents iteratively *divide* and *coordinate* until finding a division in which they agree on their solutions to their individual subproblems.

In summary, the agents running a DaC algorithm explore the space of divisions of a DCOP by repeating both stages until finding an agreement:

- a *divide* stage, in which each agent: (i) updates its local subproblem by employing the information about neighbouring subproblems gathered via *coordination*; and (ii) solves its updated local subproblem, computing its local optimal solution.
- a *coordinate* stage, in which each agent exchanges coordination information with its neighbours about their conflicts.

As discussed above, an important feature of the DaC approach is the requirement that the local subproblems at each *divide* stage compose the original DCOP. In the next section, we formally define what we understand by a valid division of a DCOP. Moreover, we also set the foundations of DaC by: (i) showing that local suproblems can be used to bound optimal solutions; and (ii) proving that DaC agreements stand for optimal solutions.

5.1.2 Divide-and-Coordinate: formal foundations

In this section we formalise the concept of valid division and its value and the two properties that relate the value of a division with the solution of the global DCOP problem.

The DaC framework requires that any division of a DCOP into local subproblems can be merged to recover the original relations. In such case, we say that the subproblems are a *valid* division of the DCOP, which we formalise as follows.

Definition 17 (Valid division)**.** *Given a DCOP Φ, a set of m subproblems $\{\Phi^s = \{\mathcal{X}^s, \mathcal{D}^s, \mathcal{R}^s\} | s = 1, \ldots, m\}$ is a valid division of Φ if its objective function, R, can be rewritten as the sum of the objective functions of the individual subproblems, namely:*

$$R(d) = R^1(d_1) + \ldots + R^m(d_m) \tag{5.1}$$

where R^s is the objective function for subproblem Φ^s, and d_s is the projection of d over $\mathcal{X}^s$, namely the variables of Φ^s.

For example, in figure 5.2(a) the set of subproblems created by agents during the *divide* stage are a valid division of the original DCOP with objective R iff: $R(x_1, x_2, x_3) = R^1(x_1, x_2, x_3) + R^2(x_1, x_2) + R^3(x_1, x_3)$. It is easy to check that combining the relations splitted in different subproblems composes the original relations in the DCOP in figure 5.1.

When creating a division we are interested on dividing the problem into simpler subproblems that are computationally tractable, and hence, can be solved by individual agents. For completeness, next we formalise a set of divisions for which a binary DCOP is divided into tree structure subproblems, and therefore they are computationally tractable (Petcu and Faltings, 2005b). .

Definition 18 (Tractable Division)**.** *Let $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a binary DCOP. We define a tractable division of Φ as a division $\{\Phi^1, \ldots, \Phi^m\}$, where $m = |\mathcal{X}|$, and each subproblem $\Phi^i = \{\mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i\}$ is defined as:*

$$\mathcal{X}^i = \{x_i\} \cup \{x_j \quad \forall x_j \in N(x_i)\}, \tag{5.2}$$

$$\mathcal{D}^i = \mathcal{D}_{\mathcal{X}^i}, \text{ and} \tag{5.3}$$

$$\mathcal{R}^i = \{r^i_i = \alpha_i \cdot r_i\} \cup \{r^i_j = (1-\alpha_i) \cdot r_j \quad \forall x_j \in N(x_i)\} \cup \{r^i_{ij} = \alpha_{ij} \cdot r_{ij} \quad \forall r_{ji} \in \mathcal{R}\} \cup \{r^i_{ji} = (1-\alpha_{ij}) \cdot r_{ji} \quad \forall r_{ji} \in \mathcal{R}\} \tag{5.4}$$

where $N(x_i)$ stands for variable x_i's neighbours in the constraint graph and α_{ij}, α_i are a real constant $0 \leq \alpha_i, \alpha_{ij} \leq 1$.

For example, in figure 5.2(a) the set of subproblems created by agents during the *divide* stage corresponds to a tractable division as described above when setting $\alpha_{ij} = \frac{1}{2} \ \forall r_{ji} \in \mathcal{R}$ and $\alpha_i = 1 \ \forall r_i \in \mathcal{R}$. In the tractable division above each subproblem Φ^i is defined over variable x_i and its neighbours $N(x_i)$. Thus, a_1's subproblem is composed of its variable x_1 and the variables of a_2, x_2, and a_3, x_3 Moreover, Φ^i is assigned the full unary relationship for variable x_i, and a α_{ij} portion of every binary relation involving x_i and a neighbour x_j. Hence, a_1's subproblem includes the unary relation over its variable x_1 and one half of the binary relations shared with a_2, $\frac{1}{2} \cdot r^{12}$, and $a3$, $\frac{1}{2} \cdot r^{13}$.

Given a valid division of a DCOP we assess its value as follows.

Definition 19 (Value of a division)**.** *Given a division $\{\Phi^s | s = 1, \ldots, m\}$ of a DCOP Φ, the value of the division is the sum of solutions of individual subproblems, namely:*

$$\sum_{s=1}^{m} R^s(x^{*,s}),$$

where $x^{*,s}$ stands for the local optimal solution of subproblem Φ^s, namely the assignment that maximises R^s.

In figure 5.2(a) the value of the division is 20 after adding: 5 (a_2' local optimum), 10 (a_1' local optimum), and 5 (a_3' local optimum).

Given the definitions of division and its value, we are ready to state two propositions that relate the value of a valid division with the value of the optimal solution.

Proposition 3. *Given a DCOP Φ with objective function R, the value of a division $\{\Phi^s | s = 1, \ldots, m\}$ of Φ is an upper bound on the value of its optimal solution, namely $R(x^*) \leq R^1(x^{*,1}) + \ldots + R^m(x^{*,m})$.*

Proof. We prove this by contradiction. Assume that there is an assignment $d \in \mathcal{D}$ whose value is greater for Φ than the value for some division $\Phi^1, \ldots, \Phi^m$ of Φ, that is $(R(d) = R^1(d_1) + \ldots + R^m(d_m)) > R^1(x^{*,1}) + \ldots + R^m(x^{*,m})$. This implies a contradiction since at least some function $R^s \in \{R^1, \ldots, R^m\}$ evaluated at d_s (the projection of d over the variables in Φ^s) should be greater than the value of its optimal solution $R^s(x^{*,s})$. □

Proposition 3 states that the value of any DCOP division is an upper bound, namely ub, on the value of its optimal solution. In figure 5.2(a) the value of the division, 20, is greater than 15, the value of the optimum of the corresponding DCOP in figure 5.1.

Proposition 4. *Given a DCOP Φ and a division $\{\Phi^s | s = 1 \ldots m\}$, if the solutions of all individual subproblems assign the very same value to each variable in $\mathcal{X}$, then this assignment is the optimal solution of Φ. In this case, the upper bound, ub, of proposition 3 is met with equality, $R(x^*) = R^1(x^{*,1}) + \ldots + R^m(x^{*,m})$.*

Proof. Assume that the optimal solutions $x^{*,1} \ldots x^{*,m}$ of the individual subproblems of a division $\{\Phi^s | s = 1 \ldots m\}$ of Φ assign the same value to each variable in $\mathcal{X}$. Let $d = x^{*,1} \cap \ldots \cap x^{*,m}$ be the values that individual subproblems assign to variables in $\mathcal{X}$. By proposition 3, we know that the value of any DCOP solution cannot be greater than the value of any of its divisions. Thus, the value of any other solution $d' \in \mathcal{D}$ of Φ is lower than the value of d, and hence d is the optimal solution of Φ. □

Figure 5.2(d) depicts a valid division of the DCOP in figure 5.1 in which all subproblems assign the very same value to each variable. Notice that the value of this division, 15, is equal to the value of the optimal solution in figure 5.1. Moreover, the joint solution on which agents agree, namely on setting x_1 to 1 and x_2, x_3 to 0, stands for its optimal solution.

The DaC approach founds on propositions 3 and 4. On the one hand, they motivate that DaC agents explore the space of divisions of a DCOP to find the one on which individual subproblems agree. As stated by proposition 4, such joint solution is an optimal solution. Moreover, it motivates that, in case of conflicts, agents exploit their local solutions to generate solutions as close to an agreement (to the optimum) as they can. We shall refer to such solutions as *candidate* solutions. For example, in the division of figure 5.2(a), agent a_1 can decide to set its variable x_1 to 1 because its two neighbours a_2 and a_3 agree on such assignment, although a_1 does not (sets x_1 to 0). In such case, a candidate solution $\mathbf{x}^{Cad}$ is composed from individual assignments: agent a_1 sets x_1 to 1, agent a_2 sets x_2 to 0, and agent a_3 sets x_3 to 0. On the other hand, the upper

bound ub of proposition 3 allows agents to provide per-instance quality guarantees over their (candidate) solutions. As stated by proposition 3, the value of any solution, and in particular of any *candidate* solution $\mathbf{x}^{Cad}$, is bounded by the value of any division. Thus, agents can assess a relative error bound δ for any x^{Cad} as $\frac{R(x^{Cad})}{ub}$. In figure 5.2(a), agents would bound the error of a candidate solution, x^{Cad}, with the value of the division, 20, assessing a relative error bound of $\delta = (15/20)$.

DaC operations	Description
Information exchange	selection of information to exchange with neighbours about local subproblem.
Subproblem update	Local update of subproblem based on utilities exchanged with neighbours.
Candidate solutions generation	Generation of candidate solutions close to an agreement as possible.

Table 5.1: Fundamental operations of the DaC framework.

Observe that the DaC framework does not constraint some fundamental operations such as how agents: (i) assess the information exchanged during the *coordinate* stage; (ii) use such information to update their subproblems during the *divide* stage; and (iii) generate *candidate* solutions. Particular implementations of these operations lead to different DaC algorithms. Table 5.1 summarises the fundamental operations that characterise the family of DaC algorithms. Next, in section 5.2, we propose a generic DaC algorithm, a DCOP incomplete algorithm that formalises the main DaC approach described in this section based on such operations.

5.2 A generic DaC algorithm

In this section we define a generic DaC algorithm that formalises the main operations of any DaC algorithm. Algorithm 5 outlines the pseudocode for the algorithm whose operation is divided into five stages: initialization, divide, coordinate, update bounded anytime solution, and termination. In what follows we detail each of these phases using the trace in figure 5.2 of a run over the DCOP of figure 5.1.

Initialization (lines 1-2). During the initialization phase agents create an initial division of the original DCOP into subproblems. Here we propose to start as initial division, the tractable division defined in section 5.1.2 when setting $\alpha_{ij} = \frac{1}{2}\ \forall r_{ji} \in \mathcal{R}$ and $\alpha_i = 1\ \forall r_i \in \mathcal{R}$

Thus, each DaC agent employs equations 5.2- 5.4 to create its initial tractable subproblem Φ^i (*createInitialSubproblem* function, line 2 in Alg. 5). Figure 5.2(a) shows the three initial subproblems (Φ^1, Φ^2, Φ^3) created by agents for the DCOP in figure 5.1. Thus, for instance, agent a_1 creates its local problem $\Phi^1 = \langle \mathcal{X}^1, \mathcal{D}^1, \mathcal{R}^1 \rangle$ for its variable x_1, where: (1) $\mathcal{X}^1 = \{x_1, x_2, x_3\}$ is composed of x_1 and its neighbours in the constraint graph; (2) $\mathcal{D}^1$ is the joint domain space for the variables in $\mathcal{X}^1$; and (3) $\mathcal{R}^1$ contains r_1, the unary relation

for x_1, and a half of each binary relation involving x_1, namely $\frac{1}{2} \cdot r_{13}$ and $\frac{1}{2} \cdot r_{23}$.

Divide (lines 4-7) . During the *divide* stage, each agent: (i) updates its local problem (*updateSubproblem* function, line 6); and (ii) subsequently solves it (*solveSubproblem* function, line 7). As explained in section 5.1, DaC agents must update their subproblem, by exchanging local utilities, such that the resulting subproblems after the update are still a valid division of the DCOP. Thus, any update of utilities in one subproblem during the *divide* stage, need to be counterbalanced by other subproblems in order to keep a valid division. As example, observe the valid division with agreement in figure 5.2(d). In this division agent a_1 have modified its local utility for variable x_2 with respect to the initial division of figure 5.2(a). Thus, a_1 has incorporated by means of relation r_2^1 a fraction of the cost to set x_2 to 1, $r_2^1(x_2 = 1) = -3$. Moreover, this update has been counterbalanced by a_2 which has modified its local relation over x_2, r_2^2, changing accordingly its utility to set x_2 to 1 from -10 to -7. In DaC we formalise these updates and the corresponding counterbalances of utilities by introducing a set of utility relations that we shall refer to as *coordination relations*. Hence, to update its local subproblem, each agent a_i assesses a set of coordination relations, namely $\{\Delta\}_i$, based on the information exchanged with its neighbours when coordinating. The set of coordination relations $\{\Delta\}_i$ is composed of:

- a unary relation Δ_j^i for each of the variables of its neighbours $x_j \in N(x_i)$. Δ_j^i quantifies how much agent a_i must change its utility for x_j to agree with a_j's assignment.
- a binary relation Δ_{ij} for each of the variables of its neighbours $x_j \in N(x_i)$. Δ_{ij} quantifies how much agent a_i must change its utility for the joint assignment of its variable, x_i, and the neighbour's variable, x_j, to agree with a_j's assignment.
- a unary relation $\bigtriangledown_i$ for its own variable x_i to counterbalance the utility updates of its neighbours. Hence, $\bigtriangledown_i = -\sum_{x_j \in N(x_i)} \Delta_i^j$.

Figure 5.2(c) shows the coordination relations assessed for each agent's subproblem. Thus, a_1 assesses a coordination relation over x_2, Δ_2^1, to quantify the change of utility in its local subproblem to agree with a_2. Thus, $\Delta_2^1(x_2 = 1) = -3$. Similarly, a_2 assesses a coordination relation $\bigtriangledown_2$ to counterbalance the utility update of a_1. Thus, $\bigtriangledown_2(x_2 = 1) = 3$.

Each agent a_i uses these coordination relations to update its local subproblem Φ^i (in *updateSubproblem* function, line 6 in Alg. 5), namely $\Phi^i = \langle \mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i \cup$

$\{\Delta\}_i\rangle$. Thus, the objective function of the updated subproblem is defined as:

$$\begin{aligned} R^i(d) = \; & r_i^i(d_i) + \bigtriangledown_i(d_i) \\ & + \sum_{x_j \in N(x_i)} \Delta_j^i(d_j) \\ & + \sum_{r_{ij}^i \in \mathcal{R}^i} (r_{ij}^i(d_i, d_j) - \Delta_{ij}(d_i, d_j)) \\ & + \sum_{r_{ji}^i \in \mathcal{R}^i} (r_{ji}^i(d_j, d_i) + \Delta_{ji}(d_j, d_i)) \end{aligned} \tag{5.5}$$

where d is an element of the joint domain space $\mathcal{D}_{\mathcal{X}^i}$ and d_i, d_j are the values assigned by d to x_i and x_j variables respectively.

Figure 5.2(c) shows the resultant agents' subproblems after updating from the initial division of figure 5.2(a). Agent a_2 includes: (i) a unary coordination relation $\bigtriangledown_2$ over its variable x_2; (ii) a unary coordination relation Δ_1^2 over its neighbour's variable x_1; and (iii) a binary coordination relation Δ_{12}, over its variable x_2 and its neighbour's variable x_1.

Agent update coordination relations at each *divide* stage. The `updateCoordinationRelations` function (line 5) specifies how each agent a_i uses the information gathered from its neighbours to assess its coordination relations ($\{\Delta\}_i$). As specified in table 5.1 the local update of subproblems varies among different DaC algorithms. Consequently, the implementation of the `updateCoordinationRelations` function (line 5) is left unconstrained.

After updating subproblems, each agent a_i solves its new subproblem Φ^i to obtain its local optimal solution ($x^{*,i}$) along with its value ($R^i(x^{*,i})$) (function *solveSubproblem*, line 6). For binary DCOPs, initial subproblems created during the *initialization* step are acyclic. Because equation 5.5 does not change the graph structure a subproblem, all subproblems remain acyclic after the update. Thus, agents' local subproblems in both the initial division of figure 5.2(a) and after the updating in figure 5.2(c) are acyclic. To solve its acyclic subproblem each agent can use any of the existing solvers in the literature, like for example the Action-GDL algorithm introduced in chapter 4, which allows to optimally solve a subproblem in linear time.

Coordinate (lines 8-12). Recall that each DaC agent updates its coordination relations in each *divide* stage with the aim of overcoming the conflicts with their neighbours. However, to rationally update their subproblems, agents need information about their neighbours local's subproblems. This information is exchanged among agents during the *coordinate* stage. During a *coordinate* stage, each agent a_i exchanges a coordination message η_{ij}, with each one of its neighbours x_j regarding the variables they share.

Coordination messages contain this information about agents' local subproblems. Then, each agent uses these coordination messages to update its coordination relations during the next *divide* stage. Figure 5.2(b) shows the messages exchanged

Algorithm 5 DaC(Φ)

Each agent a_i runs:

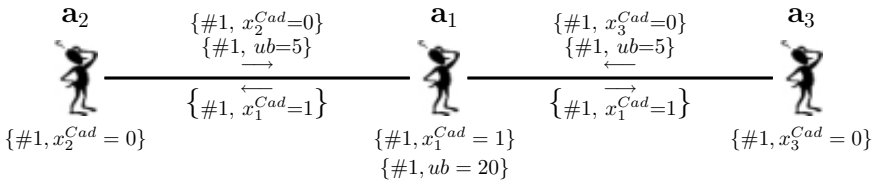
1: $\delta \leftarrow 0$; $x_i^{DaC}, x_i^{Cad} \leftarrow \emptyset$; $\{\Delta\}_i \leftarrow \emptyset$
2: $\Phi_i^0 \leftarrow$ createInitialSubproblem();
3: **repeat**
4: **/* Divide stage */**
5: $\{\Delta\}_i \leftarrow$`updateCoordinationRelations`();
6: $\Phi^i \leftarrow$ updateSubproblem$(\Phi^i, \{\Delta\}_i)$;
7: $(x^{*,i}, R^i(x^{*,i})) \leftarrow$ solveSubproblem(Φ^i);
8: **/* Coordinate stage */**
9: **for** $x_j \in Neighbours(x_i)$ **do**
10: $\eta_{ij} \leftarrow$ `wrapCoordinationInfo`();
11: $\eta_{ji} \leftarrow$ exchangeCoordinationInfo(η_{ij});
12: **end for**
13: **/*Update bounded anytime solution*/**
14: $x_i^{Cad} \leftarrow$ `generateCandidateSolution`();
15: **if** betterBoundOrSolutionAvailable$(\{\eta\})$ **then**
16: Update (δ, x_i^{DaC}) if applies
17: **end if**
18: **until** termination condition satisfied
19: **return** $\langle x_i^{DaC}, \delta \rangle$

during the coordinate stage for the initial division in figure 5.2(a). Thus, agent a_1 will send two coordination messages, one to agent a_2, namely η_{12}, and one to agent a_3, namely η_{13}. Thus, after the initial division of figure 5.2(a), agent a_1 needs to exchange some local information (e.g. its local solution) with a_2 that allows them to detect their conflict over x_2 and to assess the utilities to exchange in order to overcome such conflict.

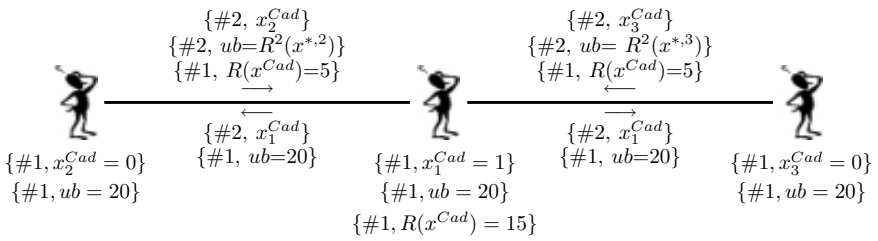
Recall that, as summarised in table 5.1, the information that agents exchange about their conflicts is not specified by the DaC framework. Thus, `wrapCoordinationInfo` function (line 10) assessing a coordination message η_{ij} (from a_i to a_j) is since each DaC algorithm must provide its particular implementation.

Update bounded anytime solutions (lines 13-17). During this stage, each agent assesses the required information to be able to return a bounded anytime solutions.

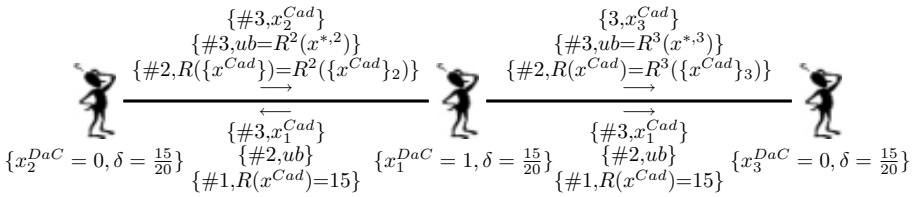
To return a bounded anytime solution each agent must: (i) generate a *candidate* solution; and (ii) assess the value of the *candidate* solution and the bound of proposition 3 (section 5.1.2). On the one hand, as explained in section 5.1.2, agents generate, during each *divide* and *coordinate* iteration, a candidate solution as close to the agreement as they can by exploiting agents' local solutions. Hence, after each *coordinate* stage, each agent a_i generates a *candidate* solution for its variable x_i (x_i^{Cad} at line 14) considering its neighbours' solutions. In that way, the candidate solution x_i^{Cad} does not have to be the same as $x_i^{*,i}$, the value



(a) Information exchanged during the 1st coordination stage.



(b) Information exchanged during the 2nd coordination stage.



(c) Information exchanged during the 3rd coordination stage.

Figure 5.3: Information exchanged between agents to update their bounded anytime solutions during the three coordination stages that follow the division stage of figure 5.2(a). # stands for the iteration number of the corresponding information.

that maximises a_i's subproblem. Thus, when a_i exchanges messages with its neighbours to coordinate, it also includes the candidate solution for variable x_i. Figure 5.3(a) shows the candidate solutions exchanged for the initial division of figure 5.2(a). For instance, a_2 sends $x_2^{Cad} = 0$ to a_1 at the first iteration.

An agent can use different strategies to generate its candidate solutions. Indeed, as summarised in table 5.1, the DaC framework does not specify the particular strategy that agents use to generate candidate solutions. Hence, the `generateCandidateSolution` function (line 14) is defined a general method. In what follows we propose two strategies to generate candidate solutions:

- **Majority rule.** Each agent a_i selects as a candidate solution for its variable x_i the value on which most agents agree. Following this strategy, for the initial division in figure 5.2(a), agent a_1 would select $x_1^{Cad} = 1$ because both a_2 and a_3 assigned 1 to x_1, despite a_1's optimal solution being $x_1^* = 0$.

- **Conditioned majority rule.** Each agent a_i selects as a candidate solution for x_i the value on which most agents agree when setting the remaining variables of their subproblems to the previous candidate solution. Recall that, during the *divide* stage each agent a_i calculates its subproblem's local solution. Besides that, when using the conditioned majority rule, it also computes for each variable $x_j \in \mathcal{X}^i$ the value that maximises R^i after setting each variable $x_k \in \mathcal{X}^i, k \neq i \; k \neq j$ to x_k^{Cad}. Then, agents exchange these alternative optimal solutions to generate the candidate solution on which most agents agree.

On the other hand, to bound the error of the candidate solution, agents need to assess: (i) the value of the candidate solution, $R(x^{Cad})$; and (ii) the value of the division. Agents need to coordinate to assess these values because each agent a_i, at each iteration, only knows the value of its local solution $R^i(x^{*,i})$ and the local value for the candidate solution $R^i(\{x^{Cad}\}_i)$, where $\{x^{Cad}\}_i$ contains the candidate solution for x_i, namely x_i^{Cad}, and the candidate solutions for each of its neighbouring variables x_j, namely x_j^{Cad}.

Thus, agents need a distributed protocol that allows them to calculate these aggregations of data and synchronize their bound and anytime solution updates. There are multiple (Zivan, 2008; Katsutoshi Hirayama, 2009) protocols available in the literature to perform this task. We chose the one detailed in Zivan (2008) because it requires only small (linear) additional space per agent and no additional messages (agents use the coordination messages to propagate information). In what follows we detail the implementation of this protocol.

In this protocol, agents are initially arranged on a directed tree. For example, figure 5.3 (a) shows a directed tree arrangement for the DCOP subproblems of figure 5.2(a), where agent a_1 is the root node with children a_2 and a_3. For each value to be aggregated, each agent: (i) receives some data from its children; (ii) aggregates these data and sends the results to its parent; (iii) receives the aggregated value from its parent; and finally (iv) relays this value to its children. These operations, which are interleaved with the DaC coordination messages, introduce little computation overhead. Figure 5.3 (a)-(c) shows the information exchanged between agents in figure 5.2(a) for this protocol. During the first round in figure 5.3(a), a_2 and a_3 start the aggregation process to calculate the upper bound ub by propagating their local optimal values for the initial division, namely $R^2(x^{*,2}) = 5$ and $R^3(x^{*,3}) = 5$, to a_1. Next, during the second round in figure 5.3(b), a_2 and a_3, calculate their local values for the candidate solution, namely $R^2(x^{Cad}) = 5$ and $R^3(x^{Cad}) = 5$, and send these data to its parent a_1. Agent a_1 aggregates the upper bound received from a_2 and a_3 for the first iteration together with the value of its local solution $R^1(x^{*,1}) = 10$. Then, as a root, a_1 sends the value of the upper bound $ub = 20$ to its children. Finally, in the third round in figure 5.3(c), agent a_1 aggregates the values of the candidate solutions from a_2 and a_3 along with its local value for its candidate solution, namely $R^{Cad}(\{x\}_i^{Cad}) = 5$. Then, as a root agent, a_1 sends the value of this aggregation $R(x^{Cad}) = 15$ back to its children a_2 and a_3. At this point, all agents have received the information

related to the aggregated data of the first iteration. When all agents have received the information related to the aggregated data for an iteration (e.g the value of the bound and of the candidate solution), they use it to update the bound (lines 14-16) and the anytime solution (lines 17-19), if applies. Because the aggregation process needs some message cycles to complete, agents will not have the actual anytime solution during the first DaC iterations. Thus, during this initial phase, agents simply return the latest generated candidate solution without giving any guarantee on its quality. Thus, each agent a_i updates the bounded anytime solution x_i^{DaC} for its variable x_i as well as the value of the bound $\delta = \frac{R(x^{DaC})}{ub}$. In figure 5.3(c), agent a_1 updates the DaC solution for its variable to $x_1^{DaC} = 1$ and its bound to $\delta = \frac{15}{20}$.

Notice that the time and space requirements for each agent are linear to the height of the chosen tree.

Termination conditions (line 18). At each iteration of the algorithm, each agent checks if some termination condition is satisfied. Typical termination conditions for DaC are: (i) the bounded anytime solution has enough quality (or it is the optimal); or (ii) the number of current iterations exceeds a maximum.

In summary, the generic DaC algorithm introduced above establishes a novel family of DCOP incomplete algorithms that can return solutions with per-instance quality guarantees. Regarding its communication complexity, at each iteration of the algorithm, each agent exchanges a message with each one of its neighbours in the constraint graph. Therefore, the number of messages exchanged per iteration is $2 \cdot |E|$, where E is the set of edges of the constraint graph. The size of these messages depends on the information exchanged about the agent's conflicts and, therefore, it will depend on the realisation that each DaC algorithm does of the `wrapCoordinationInfo` function. The data to calculate bounds and evaluate candidate configurations is linear to the height of the communication tree. Regarding its computational complexity, each agent at each iteration: (i) updates the coordination relations; (ii) creates its own subproblem in parallel with the rest of agents; and (iii) solves a tree structure subproblem. Steps (ii) and (iii) are common for all DaC algorithms and require a number of operations linear to the size of the local relations. In contrast, step (i) depends on the particular realisation that each DaC algorithm does of the function `updateCoordinationRelations`.

In the next sections we introduce two particular DaC algorithms, particular realisations of the generic DaC algorithm that explore different information and strategies to reach an agreement.

5.3 DaCSA: Divide and Coordinate Subgradient Algorithm

In this section we formulate the so-called Divide And Coordinate Subgradient Algorithm (DaCSA), a particular computational realisation of the DaC approach where agents: (i) coordinate by exchanging their local solutions as information about their

conflicts; and (ii) in case of conflict, exchange utilities for their local solutions with a neighbour. DaCSA has its formal foundations on Lagrangian dual decompositions and subgradient methods (Bertsekas, 2007). Next, in section 5.3.1 we provide the formal foundations of DaCSA, while in section 5.3.2, we provide its algorithmic details.

5.3.1 Formal foundations

To build a computational realization of the DaC approach to solve a DCOP we must define: (i) what information agents exchange about local subproblems during the *coordinate* step; and (ii) how to use that information to create, at each *divide* step, a valid tractable division whose subproblems' solutions are closer to an agreement. With this aim we propose to use Lagrangian dual decomposition along with subgradient methods, both well-known techniques in optimization with strong theoretical properties (refer to (Bertsekas, 2007), section 6.4).

Let Φ be a binary DCOP. To apply duality we need to formalize Φ as a binary linear program (LP). Let $\{\Phi^i | i = 1 \dots m\}$ be an initial division of Φ as defined in section 5.2 by equations 5.2-5.4. Then, for each subproblem Φ^i we define the following binary variables:

- $x^i_{j;l}$, that takes on value 1 when variable x_j in subproblem Φ^i takes on value l.
- $x^i_{ij;kl}$, that takes on value 1 when variables x_i,x_j in subproblem Φ^i take on values k and l respectively.

Formally, the set of binary variables of subproblem Φ^i is given by:

$$\begin{aligned} X^i_{LP} =& \{x^i_{i;k} : \forall k \in \mathcal{D}_i\} \cup \{x^i_{j;l} : \ \forall x_j \in N(x_i) \ \ \forall l \in \mathcal{D}_j\} \cup \\ & \{x^i_{ij;kl} : \ \forall r^i_{ij} \in \mathcal{R}^i \ \forall k \in \mathcal{D}_i \ \forall l \in \mathcal{D}_j\} \cup \{x^i_{ji;lk} : \ \forall r^i_{ji} \in \mathcal{R}^i \ \forall l \in \mathcal{D}_j \ \forall k \in \mathcal{D}_i\} \end{aligned}$$

With this set of variables we can express the objective function for subproblem Φ^i as:

$$\begin{aligned} R^i_{LP}(X^i_{LP}) = & \sum_{k \in \mathcal{D}_i} x^i_{i;k} \cdot r^i_i(k) + \sum_{r^i_{ij} \in \mathcal{R}^i} \sum_{k \in \mathcal{D}_i} \sum_{l \in \mathcal{D}_j} x^i_{ij;kl} \cdot r^i_{ij}(k,l) \\ & + \sum_{r^i_{ji} \in \mathcal{R}^i} \sum_{l \in \mathcal{D}_j} \sum_{k \in \mathcal{D}_i} x^i_{ji;lk} \cdot r^i_{ji}(l,k) \end{aligned}$$

Then, solving Φ amounts to solving the following LP:

$$\max_{\{X^i_{LP}\}} \sum_{i=1}^{|\mathcal{X}|} R^{iLP}(X^i_{LP}) \tag{5.6}$$

subject to the following constraints ($\forall i \in \{1, \ldots, |\mathcal{X}|\}$):

(C1) A unique value is assigned to each variable:

$$\begin{aligned} &\sum_{k \in \mathcal{D}_i} x^i_{i;k} = 1 && \forall x_i \in \mathcal{X}^i \\ &\sum_{k \in \mathcal{D}_i} \sum_{l \in \mathcal{D}_j} x^i_{ij;kl} = 1 && \forall r^i_{ij} \in \mathcal{R}^i \\ &\sum_{l \in \mathcal{D}_j} \sum_{k \in \mathcal{D}_i} x^i_{ji;lk} = 1 && \forall r^i_{ji} \in \mathcal{R}^i \end{aligned} \tag{5.7}$$

(C2) A variable is assigned the very same value in all relations:

$$\begin{aligned} &\forall r^i_{ij} \in \mathcal{R}^i : \quad x^i_{i;k} = \sum_{l \in \mathcal{D}_j} x^i_{ij;kl} \ \ \forall k \in \mathcal{D}_i, \quad x^i_{j;l} = \sum_{k \in \mathcal{D}_i} x^i_{ij;kl} \ \ \forall l \in \mathcal{D}_j \\ &\forall r^i_{ji} \in \mathcal{R}^i : \quad x^i_{i;k} = \sum_{l \in \mathcal{D}_j} x^i_{ji;lk} \ \ \forall k \in \mathcal{D}_i, \quad x^i_{j;l} = \sum_{k \in \mathcal{D}_i} x^i_{ji;lk} \ \ \forall l \in \mathcal{D}_j \end{aligned} \tag{5.8}$$

(C3) Subproblems agree on variables' values:

$$\begin{aligned} &x^j_{i;k} = x^i_{i;k} && \forall x_j \in N(x_i) \ \ \forall k \in \mathcal{D}_i \\ &x^j_{ij;kl} = x^i_{ij;kl} && \forall r_{ij} \in \mathcal{R}^i \ \ \forall k \in \mathcal{D}_i \ \ \forall l \in \mathcal{D}_j \end{aligned} \tag{5.9}$$

Notice that the sets of constraints (C1) and (C2) ensure consistency in assignments inside each subproblem, whereas the set of constraints (C3) ensures consistency between subproblems. Then, solving Φ amounts to solving the following Lagrangian dual problem:

$$\min_{\{\lambda\}} \max_{\{X^i_{LP}\}} R_{DUAL}(\{X^i_{LP}\}; \{\lambda\}) = \min_{\{\lambda\}} \sum_{i=1}^{|\mathcal{X}|} \max_{X^i_{LP}} R^i_{DUAL}(X^i_{LP}; \{\lambda\}) \tag{5.10}$$

where the objective function R^i_{DUAL} corresponding to subproblem Φ^i is defined as:

$$\begin{aligned} R^i_{DUAL}(X^i_{LP}; \{\lambda\}) = \ & \sum_{k \in \mathcal{D}_i} x^i_{i;k} \cdot (r^i_i(k) - \sum_{x_j \in N(x_i)} \lambda^j_{i;k}) \\ & + \sum_{x_j \in N(x_i)} \sum_{l \in \mathcal{D}_j} x^i_{j;l} \cdot \lambda^i_{j;l} \\ & + \sum_{r^i_{ij} \in \mathcal{R}^i} x^i_{ij;kl} \cdot (r^i_{ij}(k,l) - \lambda_{ij;kl}) \\ & + \sum_{r^i_{ji} \in \mathcal{R}^i} x^i_{ji;kl} \cdot (r^i_{ji}(k,l) + \lambda_{ji;kl}) \end{aligned} \tag{5.11}$$

subject to the set of constraints (C1) and (C2).

Thus, given set of Lagrange multipliers, each agent a_i can solve its subproblem independently, by maximising the R^i_{DUAL} objective function. Moreover, subproblems generated by equation 5.11 conform a valid division of the original DCOP. Hence, propositions 3 and 4 apply to the subproblems that compose the dual problem of equation 5.10 so that: (i) if all subproblems agree on the value of shared variables then these values

are the DCOP solution; and (ii) the sum of the value of the solutions of individual subproblems provides a bound on the quality of the solution of the original problem. Furthermore, there is a direct correspondence between the dual function of a subproblem Φ^i, R^i_{DUAL}, and the updated objective function of subproblem Φ^i, R^i, as introduced in section 5.2 (equation 5.5) for the generic DaC algorithm. Concretely, there is a direct correspondence between both equations when using the following mapping between coordination relations and Lagrange multipliers:

$$\begin{array}{ll} \Delta^j_i(k) = \lambda^i_{j;k} & \forall k \in \mathcal{D}_j \\ \Delta_{ij}(k,l) = \lambda_{ij;kl} & \forall k \in \mathcal{D}_i \forall l \in \mathcal{D}_j \\ \Delta_{ji}(k,l) = \lambda_{ji;kl} & \forall k \in \mathcal{D}_j \forall l \in \mathcal{D}_i \end{array} \tag{5.12}$$

Thus, in DaCSA the coordination relations used to update subproblems by the `updateSubproblem` function (algorithm 5) correspond to the Lagrange multipliers, as stated by the mapping in equation 5.12.

An important issue on the dual formulation is how to assess Lagrange multipliers that minimize equation 5.10, namely the violation of constraints, to bring solutions of dual subproblems to an agreement. With that purpose, we use the subgradient method (Bertsekas, 2007), an iterative method that allows to update Lagrange multipliers at each iteration in parallel from the solutions of neighbouring subproblems. According to subgradient methods, the set of Lagrange multipliers $\{\lambda\}$ is updated using the subgradient of the Lagrangian dual problem in equation 5.10. The subgradient of the dual problem is a vector with one component for each $\lambda^j_{i;k}, \lambda_{ij;kl} \in \{\lambda\}$. The component of the subgradient related to a Lagrange multiplier $\lambda \in \{\lambda\}$ can be assessed as the expression that multiplies λ in equations 5.11 when evaluated at R_{DUAL} optimal solution. Hence, following the subgradient method, Lagrange multipliers in each subproblem Φ^i are updated as [1]:

$$\begin{array}{ll} \lambda^j_{i;k} \leftarrow \lambda^j_{i;k} - \epsilon \cdot (x^{*,j}_{i;k} - x^{*,i}_{i;k}) & \forall k \in \mathcal{D}_i \\ \lambda^i_{j;l} \leftarrow \lambda^i_{j;l} - \epsilon \cdot (x^{*,i}_{j;l} - x^{*,j}_{j;l}) & \forall l \in \mathcal{D}_j \\ \lambda_{ij;kl} \leftarrow \lambda_{ij;kl} - \epsilon \cdot (x^{*,j}_{ij;kl} - x^{*,i}_{ij;kl}) & \forall k \in \mathcal{D}_i \; \forall l \in \mathcal{D}_j \\ \lambda_{ji;kl} \leftarrow \lambda_{ji;kl} - \epsilon \cdot (x^{*,i}_{ji;kl} - x^{*,i}_{ji;kl}) & \forall k \in \mathcal{D}_j \; \forall l \in \mathcal{D}_i \end{array} \tag{5.13}$$

where $x^{*,i}_{i;k}$ is the optimal assignment for variable $x^i_{i;k}$ in subproblem Φ^i and ϵ is a positive real step-size.

Equations in 5.13 help us realise the *divide* and *coordinate* stages of the DaC approach. On the one hand, regarding the *divide* stage, equations in 5.13 and 5.12 realise the `updateCoordinationRelations` function in the general DaC algorithm (algorithm 5) with the assessment of the coordination relations, the Lagrange multipliers. Following equations in 5.13, the value of a Lagrange multiplier is modified whenever subproblems conflict on the value assigned to a variable, and it remains unchanged if they agree. This can be interpreted as an attempt to reduce disagreement between subproblems. Moreover, the update of Lagrange multipliers considers a step-size ϵ (a positive real value) that weights the impact of disagreements on updates. The larger the

[1] Each Lagrange multiplier is initially set to 0.

Algorithm	Information exchange	Subproblem update
DaCSA	Local optimal solutions	When a neighbour disagrees on the optimal value of a variable give away some utility in favour of such value to convince him.
EU-DaC	Max-marginals	When the max-marginal of a neighbour is different from the own max-marginal, exchange utilities to get closer max-marginals.

Table 5.2: Different computational realisations of the fundamental DaC operations.

value of ϵ, the higher the impact of disagreements on Lagrange multipliers. Different types of step-size rules have been proposed for subgradient methods, each one with different guarantees and convergence rate results (refer to Bertsekas (2007), section 6.3.1 subgradient methods). On the other hand, regarding the *coordinate* stage, equations in 5.13 and 5.12 determine which coordination information is needed to update the Lagrange multipliers of a subproblem Φ^i, namely the local solution of each neighbour subproblem Φ^j over their shared variables.

In summary, this section defines the formal foundations of DaCSA that provide a particular realisation of the DaC fundamental operations summarised in table 5.1, namely: (i) the information agents exchange with its neighbours about its local subproblem; and (ii) how to update subproblems based on such information. Hence, as summarised in table 5.2, each DaCSA agent a_i: (i) coordinates by exchanging its local solutions with its neighbours; and (ii) in case of conflict with a neighbour a_j, gives away some utility for its local solution to a_j.

5.3.2 DaCSA algorithm

This section details how DaCSA realises the *divide* and *coordinate* stages of the generic DaC algorithm introduced in section 5.2. Hence, next we describe the particular realisations of DaCSA for the `updateCoordinationSubproblem` and the `wrapCoordinationInfo` procedures of algorithm 5. To illustrate the operation of DaCSA, we describe the trace of a run over the DCOP in figure 5.1 as depicted in figure 5.4.

Agents start with the initial division shown in figure 5.2(a). Because no coordination information is available, agents proceed to solve their initial subproblems without adding any coordination relation. Thus, figure 5.2(a) depicts these initial local solutions assessed by agents for their initial subproblems as well as their values. According to the DaC approach, agents' solutions may disagree after a *divide* stage, as it is the case in figure 5.2(a). Thus, agent a_1 disagrees with a_2 and a_3 on the optimal configuration of all their shared variables, namely x_1,x_2 and x_1,x_3 respectively.

Then, DaCSA agents proceed to coordinate. Function 6 outlines DaCSA's implementation of the `wrapCoordinationInfo` function. Following function 6, each DaCSA agent a_i exchanges with each one of its neighbours a_j a message η_{ij} that containing the local solutions for common variables, namely $x_i^{*,i}$ and $x_j^{*,i}$. Figure 5.4(a) shows the coordination messages that agents exchange for the initial division of figure

Function 6 $wrapCoordinationInfo(x_i^{*,i}, x_j^{*,i})$

1: $\eta_{ij} = \langle x_i^{i,*}, x_j^{i,*} \rangle$;
2: **return** η_{ij};

Function 7 $updateCoordinationRelations(\Phi^i, \{\eta\}_i)$

Agent a_i runs:

1: $\epsilon \leftarrow$ `calculateStepsize();`
2: **for** $x_j \in \mathcal{N}(x_i)$ **do**
3: **for** $\forall d_j \in \mathcal{D}_j$ /*For each value of x_j*/ **do**
4: /*Update the coordination relation for the neighbour variable*/
5: $\Delta_j^i(d_j) = \Delta_j^i(d_j) + \epsilon \cdot (\left[\eta_{ji}.x_j^{*,j} = d_j\right] - \left[x_j^{*,i} = d_j\right])$
6: **end for**
7: **for** $\forall d_i \in \mathcal{D}_i$ /*For each value of x_i*/ **do**
8: /*Update the coordination (balancing) relation for its own variable*/
9: $\bigtriangledown_i^i(d_i) = \bigtriangledown_i^i(d_i) + \epsilon \cdot (\left[\eta_{ji}.x_i^{*,j} = d_i\right] - \left[x_i^{*,i} = d_i\right])$
10: **end for**
11: **end for**
12: /*Update binary coordination relations */
13: **for** $r_{ij} \in \mathcal{R}$ **do**
14: **for** $\forall d_{ij} \in \mathcal{D}_{ij}$ /*For each value of x_i, x_j*/ **do**
15: $\Delta_{ij}(d_{ij}) = \Delta_{ij}(d_{ij}) - \epsilon \cdot (\left[\eta_{ji}.\langle x_i^{*,j}, x_j^{*,j} \rangle = d_{ij}\right] - \left[\langle x_i^{*,i}, x_j^{*,i} \rangle = d_{ij}\right])$
16: **end for**
17: **end for**
18: **for** $r_{ji} \in \mathcal{R}$ **do**
19: **for** $\forall d_{ji} \in \mathcal{D}_{ji}$ /*For each value of x_i, x_j*/ **do**
20: $\Delta_{ji}(d_{ij}) = \Delta_{ji}(d_{ij}) + \epsilon \cdot (\left[\eta_{ji}.\langle x_i^{j,*}, x_j^{j,*} \rangle = d_{ij}\right] - \left[\langle x_i^{i,*}, x_j^{i,*} \rangle = d_{ij}\right])$
21: **end for**
22: **end for**
23: **return** $\{\Delta\}_i$;

5.2(a). Thus, agent a_1 sends a message to a_2 with assignments $\{x_1^{*,1} = 0, x_2^{*,1} = 1\}$, and a message to a_3 with assignments $\{x_1^{*,1} = 0, x_3^{*,1} = 1\}$.

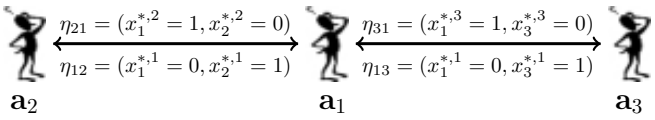
After that exchange, agents proceed to update coordination relations by means of the `updateCoordinationRelations` function, whose pseudocode is given in function 7. Thus, according to function 7 each agent assesses the Lagrange multipliers updates of equation 5.13, but in terms of the coordination relations by using the mapping in equation 5.12. Operation $[\cdot]$ stands for a function that returns 0 if the equality inside is false $[false] = 0$ and return 1 otherwise, $[true] = 1$. Following procedure 7, each agent a_i starts by calculating the step-size ϵ for that iteration according to the chosen step-size rule (line 1).

Next, each agent a_i uses the assignments received from each of its neighbours a_j in

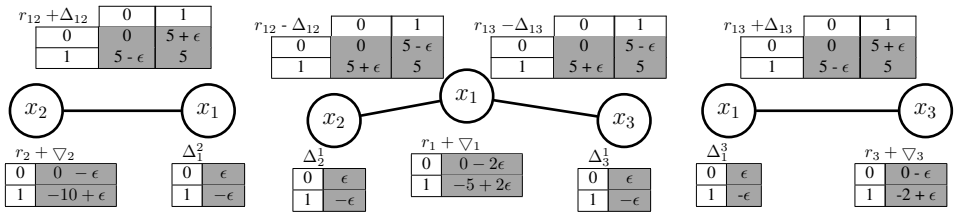
the coordination message η_{ji}, along with its local assignments, to update the coordination relations (lines 2-22). Figure 5.4(b) shows the coordination relations assessed by agents for the initial division in figure 5.2(a) after coordination.

On the one hand, each agent a_i updates for each of the variables of its neighbours $x_j \in N(x_i)$ the value of its coordination relation Δ_j^i. This quantifies how much agent a_i must change its utility for x_j to agree with a_j's solution (lines 3-6). Each agent a_i updates the value of coordination relation Δ_j^i over x_j when a_i's optimal solution differs from a_j's by: (i) increasing the utility for a_j's solution $x_j^{*,j}$ by ϵ; and (ii) decreasing the utility for its own solution $x_j^{*,i}$ by ϵ. For instance, in figure 5.4(b), a_2 will modify its utility for x_1 to agree with a_1's solution. Thus, since a_2's optimal solution for x_1 is 1, whereas a_1's optimal solution is 0, a_2 increases $\Delta_1^2(0)$ and decreases $\Delta_1^2(1)$ by ϵ. In that way, a_2 decreases the utility for its current x_1 optimal solution, whereas increments its utility for a_1's optimal solution getting closer to the agreement. In a similar way, each agent a_i updates the coordination relation over its variable x_i, $\bigtriangledown_i$, to counterbalance the utility updates of its neighbours (lines 7-10). Thus, in figure 5.4(b), a_1 decreases the utility for its variable x_1 when it is set to its optimal solution 0, whereas increases the utility for x_1 set to 1 in order to counterbalance the updates of a_2 and a_3. On the other hand, each agent a_i updates for each of its relations $r_{ij}, r_{ji} \in \mathcal{R}^i$ the value of coordination relation Δ_{ji}^i (or $-\Delta_{ji}^i$ if $j < i$) (lines 12-22). Each agent a_i updates the value of coordination relation Δ_{ji}^i when it disagrees with a_j on the joint solution of variables x_i and x_j by: (i) increasing a_j's solution for x_i, x_j by ϵ; and (ii) decreasing the value of its own solution for x_i, x_j by ϵ. Thus, in figure 5.4(b), agent a_2 uses the assignments received from a_1 for their common variables, namely $\{x_1^{*,1} = 0, x_2^{*,1} = 1\}$ to update coordination relation Δ_{12}^2 trying to decrease the disagreement with a_1. Since both agents agree that joint configurations $\{x_1 = 0, x_2 = 0\}$ and $\{x_1 = 1, x_2 = 0\}$ are not optimal, the value of the coordination relation for these configurations ($\Delta_{12}^2(0,0)$, $\Delta_{12}^2(1,1)$) remain unchanged. In contrast, agent a_2 increases the value $\Delta_{12}^2(0,1)$ in favour of a_1's optimal solution and decreases $\Delta_{12}^2(1,0)$ in detriment of its own optimal solution.

Finally, each agent a_i uses its coordination relations to update its local subproblem as specified by the *updateSubproblem* function in the generic DaC algorithm (line 6, algorithm 5). Thus, for instance, in figure 5.4(b), the new set of relations $\mathcal{R}^2$ created by a_2 is composed of: (1) the unary relation r_2 along with a $\bigtriangledown_2$ coordination relation to coordinate the x_2 assignments with a_1; (2) binary relation r_{12}^2 along with a coordination relation Δ_{12} to coordinate the $\{x_1, x_2\}$ assignments with a_1; and (4) a coordination relation Δ_1^2 to coordinate the x_1 assignments with a_1. Thereafter, each agent a_i solves its new subproblem to obtain its optimal local solution, $x^{*,i}$, along with its value $R(x_i^*)$ (line 6, algorithm 5). For instance, figure 5.4(b) shows the local solutions and values computed by each agent when setting $\epsilon = 1$. As a result, observe that agent a_1 and a_3 change their assignments with respect to the first iteration: agent a_1 sets its variable x_1 to 1, and variables x_2, x_3 to 0, whereas agent a_3 sets its variable x_3 to 1, and variable x_1 to 0. Observe that as a result of these changes, agent a_1 know agrees on the assignments of the shared variables with agent a_2. Moreover, the sum of utilities of subproblems' solutions for this division, $2 + 11 + 6 = 19$, is lower than those for the initial division, $5 + 10 + 5 = 20$. Since the sum of utilities of subproblems' solutions is an upper bound



(a) DaCSA coordinate stage.



(b) New subproblems after DaCSA divide stage when setting an stepsize ϵ to 1.

Figure 5.4: Trace of DaCSA over the DCOP initial division of figure 5.2(a).

on the quality of the optimal optimal solution, the agents obtain during this *divide* stage a tighter bound and new assignments closer to an agreement.

At the end of each iteration, the agents check if some termination condition is satisfied (line 18, algorithm 5). In figure 5.4(b), because after the first iteration agent a_1 still disagrees with agent a_3, agents would proceed to execute a new iteration of the DaCSA algorithm.

5.3.3 Complexity analysis

We can directly assess the complexity of DaCSA from the complexity of the generic DaC algorithm and from the particular DaCSA implementations of the *wrapCoordinationInfo* and *updateCoordinationRelations* functions. DaCSA realises the *wrapCoordinationInfo* procedure by assessing a message from a_i to a_j that contains the assignments for their shared variables. Therefore, the size of the coordination messages is logarithmic in the domain of variables. Moreover, each agent updates the coordination relations requiring a number of operations linear to the size of the local relations. As a result, DaCSA is a low-overhead algorithm because agents exchange a linear number of messages of linear size and perform a linear number of operations.

5.4 EU-DaC: Egalitarian Utilities Divide And Coordinate algorithm

Several works (Farinelli et al., 2008; Wainwright et al., 2005) have shown that agents obtain better solutions when they explicitly communicate the utilities of their assignments instead of their preferred assignments. Along this line, here we propose the so-called Egalitarian Utilities Divide and Coordinate algorithm (EU-DaC), a DaC algorithm that has agents coordinate by exchanging the utilities of their variables' assignments. The rationale behind EU-DaC is the following: if agents agree on the utilities of their shared variables, then they agree on their local assignments[2]. According to the DaC framework, this implies that agents have found a DCOP solution.

Next, section 5.4.1 provides the formal foundations of EU-DaC while in section 5.4.2 we provide its algorithmic details.

5.4.1 Formal foundations

Max-marginals have been widely used in the literature to compute marginal probabilities and most probable configurations in graphical models (Wainwright and Jordan, 2008; Aji and McEliece, 2000; Farinelli et al., 2008). A max-marginal utility function of a DCOP over a variable summarises the total dependency of the DCOP over such variable. More formally, a max-marginal utility function $\mathcal{U}_j^i : D_j \rightarrow \Re$ encodes for each possible value of x_j, $l \in \mathcal{D}_j$, the maximum reward of Φ^i when x_j is set to l:

$$\mathcal{U}_j^i(l) = \max_{d \in \mathcal{D}_{\mathcal{X}^i \setminus x_j}} R^i(l; d) \tag{5.14}$$

In figure 5.5 the max-marginal utilities of subproblem Φ^2 (depicted on the left of the figure) for variable x_2 ($\mathcal{U}_2^2$) are computed as:

$$\mathcal{U}_2^2(0) = \max_{d_1 \in \mathcal{D}_1} r_1(d_1) + r_{12}(d_1, 0) + r_2(0) = 5$$
$$\mathcal{U}_2^2(1) = \max_{d_1 \in \mathcal{D}_1} r_1(d_1) + r_{12}(d_1, 1) + r_2(1) = \frac{8}{3}$$

Hence, subproblem Φ^2 has a local max-marginal utility of 5 when setting variable x_2 to 0, and a local max-marginal utility of $\frac{8}{3}$ when setting it to 1.

We define the optimum of a max-marginal $\mathcal{U}_j^i$ as the value for x_j that maximises it. Formally,

$$x_j^{*,\mathcal{U}^i} = arg \max_{l \in \mathcal{D}_j} \mathcal{U}_j^i(l) \tag{5.15}$$

If $x_j^{*,\mathcal{U}^i}$ is the unique value that maximises $\mathcal{U}_j^i$, then we say that the max-marginal has a unique optimum. In figure 5.5 the max-marginal of agent a_2 for its variable x_2 has a unique optimum, namely $x_2 = 0$.
Next, we define the condition of max-marginal agreement in a DCOP division.

[2]This statement is subject to the absence of utility ties: different variable assignments have different utility values.

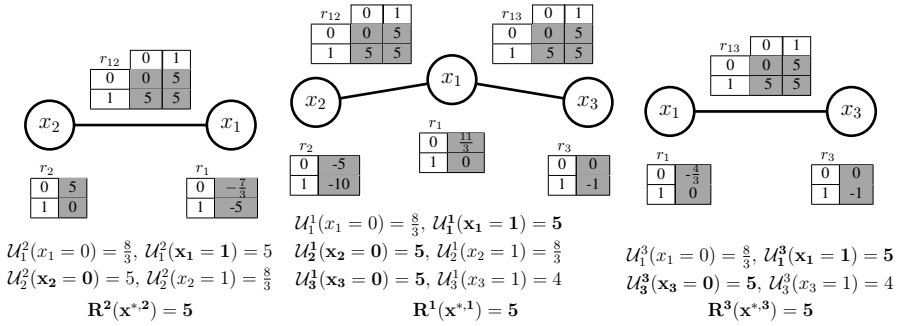


Figure 5.5: Division with max-marginals agreement for the DCOP of figure 5.1.

Definition 20. *Two subproblems Φ^s and Φ^m agree on their max-marginals when for each of its shared variables $x_i \in \mathcal{X}^s \cap \mathcal{X}^m$ the max-marginals of Φ^s and Φ^m over x_i assign the same utility for all possible values of x_i. Formally,*

$$\mathcal{U}_i^s(k) = \mathcal{U}_i^m(k) \quad \forall x_i \in \mathcal{X}^s \cap \mathcal{X}^m, \quad \forall k \in \mathcal{D}_i \tag{5.16}$$

In figure 5.5 subproblem Φ^1 and Φ^2 agree on their max-marginals because the max-marginals over each of its shared variables, namely x_1 and x_2, assign the same utility for each of the values of x_1 and x_2.

Proposition 5. *Given a DCOP Φ and a division $\{\Phi^s | s = 1 \dots m\}$, if each pair of subproblems agree on their max-marginals and each max-marginal has a unique optimum, then the solution of all individual subproblem agree on assigning the same value to each variable $x_i \in \mathcal{X}$, namely x_i^*. In this case, the value of x_i^* is equal to the value of the unique optimum that maximises the max-marginal of each subproblem.*

Proof. We prove this by contradiction. Assume that there are two subproblems Φ^s and Φ^m such that they agree on their max-marginals over a variable x_i and for which the optimal value of Φ^s for x_i, $x_i^{*,s}$, and Φ^m, $x_i^{*,m}$, differ ($x_i^{*,s} \neq x_i^{*,m}$). Because in proposition 5 each max-marginal is required to have unique optimum it implies that for Φ^m, $\mathcal{U}_i^m(x_i^{*,m}) > \mathcal{U}_i^m(x_i^{*,s})$ and for Φ^s, $\mathcal{U}_i^s(x_i^{*,s}) > \mathcal{U}_i^s(x_i^{*,m})$. This leads to a contradiction because if Φ^s and Φ^m agree on their max-marginals $\mathcal{U}_i^s(k) = \mathcal{U}_i^m(k), \forall k \in \mathcal{D}_i$. □

The EU-DaC approach founds on proposition 5. Agents to search for a division whose subproblems agree on max-marginals. Figure 5.5 shows a division of the DCOP in figure 5.1 in which agents agree on their max-marginals, where the unique optimum of each max-marginal is boldfaced. Thus, the unique optimum of x_1 in max-marginals $\mathcal{U}_1^{s=1\dots3}$ is 1, of x_2 in max-marginals $\mathcal{U}_2^{s=1,2}$ is 0, and of x_3 in max-marginals $\mathcal{U}_3^{s=1,3}$ is 0. Observe that these assignments are the optimal values of the variables in the DCOP of figure 5.1.

Notice that, a division in which subproblems agree on their max-marginals subproblems agree on their individual solutions, the other way around is not true. Figure 5.2(d)

shows an example of a division but subproblems agree on their local solutions, whereas their max-marginals differ. For example, the value of a_1's max-marginal to set x_1 to 0 is $\mathcal{U}_1^1(0) = 2.5$, whereas the value of a_3's max-marginal is $\mathcal{U}_1^3(0) = 5$. Hence, EU-DaC will converge under two different conditions, namely when it finds a division: (i) whose subproblems max-marginals agree; or (ii) whose subproblems' solutions agree.

Based on proposition 5, the goal of each EU-DaC agent when updating its individual subproblem is to converge on the max-marginal utilities of its neighbours regarding their shared variables. Hence, proposition 5 helps us to realise the *divide* stage by updating the coordination relations based on the differences between subproblems' max-marginals. On the one hand, given a division $\{\Phi^i | i = 1, \ldots, |\mathcal{X}|\}$, each agent a_i updates for each of its neighbours' variables $x_j \in N(x_i)$ a coordination relation $\triangle_j^i$ that quantifies how much utility is required to agree with a_j's max-marginal. Formally:

$$\triangle_j^i(l) = \triangle_j^i(l) + \rho \cdot \Big(\mathcal{U}_j^j(l) - \mathcal{U}_j^i(l)\Big) \quad \forall l \in \mathcal{D}_j \tag{5.17}$$

where $\rho \in [0, 1]$ is a damping parameter that weighs the magnitude of the change over the subproblem. On the other hand, each agent a_i assesses a coordination relation for its own variable x_i to counterbalance the utility updates of its neighbours, namely $\bigtriangledown_i(k) = \bigtriangledown_i(k) + \sum_{x_j \in \mathcal{X}^i, j \neq i} \triangle_j^i(k) \;\; \forall k \in \mathcal{D}_i$.

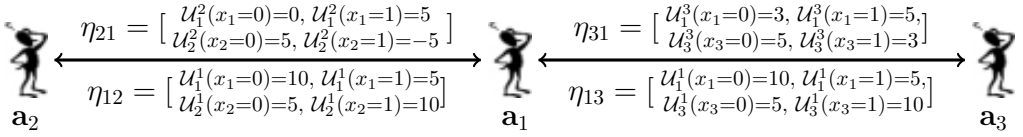
Finally, regarding the *coordinate* stage, equation 5.17 defines which coordination information is needed to update the coordination relations of a subproblem Φ^s, namely the local max-marginals of each neighbour subproblem Φ^m over each of their shared variables.

In summary, we defined the particular EU-DaC implementations of the two fundamental DaC operations, namely: (i) *what information* agents exchange about their conflicts; and (ii) *how to update subproblems* based on such information. Hence, each EU-DaC agent a_i: (i) coordinates by exchanging the individual max-marginals of each of its shared variable with its neighbours; and (ii) when max-marginals differ, exchanges utilities with its neighbours to bring their max-marginals closer.

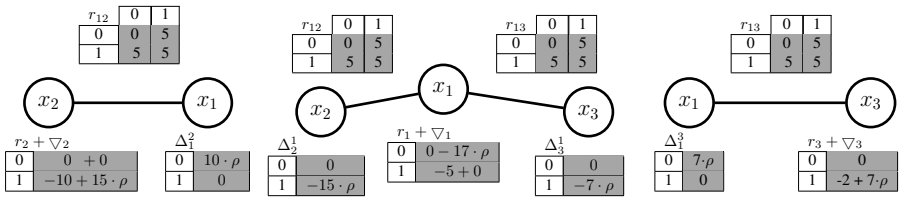
5.4.2 EU-DaC algorithm

In this section we detail how EU-DaC realises the *coordinate* and *divide* stages of the generic DaC algorithm introduced in section 5.2 based on the formal foundations established in the section above. Next, we describe how EU-DaC implements the `updateCoordinationRelations` and the `wrapCoordinationInfo` functions of the generic DaC algorithm. To illustrate the operation of the EU-DaC algorithm we describe the trace of a run over the DCOP in figure 2.1 depicted in figure 5.6.

Agents start with the initial division shown in figure 5.2(a). Similarly to DaCSA at this initial *divide* stage no coordination information is available, so agents proceed to solve their initial subproblems without adding any coordination relation. Figure 5.2(a) depicts the initial local solutions calculated by agents for their initial subproblems as well as the value of such solutions. However, in EU-DaC, each agent a_i also needs to assess the max-marginals over each variable in $\mathcal{X}^i$. It is convenient then to use a solver such as the Max-Sum algorithm (Farinelli et al., 2008), which returns, in addition to the optimal solution and its value, the max-marginal utilities over single variables



(a) EUDaC coordinate stage.



(e) New subproblems after EU-DaC divide stage and solutions obtained using a damping parameter $\rho = \frac{1}{4}$.

Figure 5.6: Trace of EU-DaC over the DCOP initial division of figure 5.2(a).

without any additional cost. Recall in figure 5.2(a) after the initial *divide* stage , agent a_1 disagrees with a_2 and a_3 on the optimal value of all their shared variables, namely x_1, x_2 and x_1, x_3 respectively.

Thus, EU-DaC agents proceed to coordinate. Function 8 implementation of the EU-DaC realisation of the `wrapCoordinationInfo` function. Following function 8, each EU-DaC agent a_i exchanges a message η_{ij} with each one of its neighbours a_j that contains the max-marginals for their common variables, namely $\mathcal{U}_i^i$ and $\mathcal{U}_j^i$. Figure 5.6(a) shows the coordination messages exchanged between EU-DaC agents during the *coordination* stage. For instance, agent a_2 sends to a_1 a message containing the max-marginal utilities for their shared variables for its initial subproblem in figure 5.2(a), namely x_1 ($\mathcal{U}_1^2$) and x_2 ($\mathcal{U}_2^2(0)$). Concretely, agent a_2 assesses its local max-marginals for its variable x_2 as:

$$\mathcal{U}_2^2(0) = \max_{d_1 \in \mathcal{D}_1} r_1(d_1) + r_{12}(d_1, 0) + r_2(0) = 5$$
$$\mathcal{U}_2^2(1) = \max_{d_1 \in \mathcal{D}_1} r_1(d_1) + r_{12}(d_1, 1) + r_2(1) = -5$$

Hence, agent a_2 reports its coordinate message to a_1, a local max-marginal utility of 5 when setting its variable x_2 to 0 ($\mathcal{U}_2^2(0) = 5$), and a local max-marginal utility of -5 when setting it to 1 ($\mathcal{U}_2^2(1) = -5$).

Once received the coordination messages from its neighbours, each EU-DaC agent assesses its coordination relations ($\{\bigtriangleup^i\}$ and $\{\bigtriangledown^i\}$) by executing the `updateCoordinationRelation` function. Function 9 outlines the pseudocode of the EU-DaC implementation of `updateCoordinationRelation` function. Following function 9, each agent a_i starts by calculating the damping parameter ρ for that

iteration (line 1). Next, each agent a_i uses the max-marginals received from each of its neighbours a_i in the coordination message η_{ij}, along with its local max-marginals, to update the coordination relations (lines 2-11). Figure 5.6(b) shows the coordination parameters assessed by EU-DaC agents for the initial division in figure 5.2(a) after coordination. Notice that, unlike DaCSA, EU-DaC agents coordinate using only coordination relations defined over single variables.

Function 8 $wrapCoordinationInfo(\{\mathcal{U}\}_i)$

1: $\eta_{ij} = \langle \mathcal{U}_i^i, \mathcal{U}_j^i \rangle$;
2: **return** η_{ij};

Function 9 $updateCoordinationRelations(\Phi^i, \{\eta\}_i)$

Each agent a_i runs:

1: $\rho \leftarrow$ `calculateDampingParameter();`
2: **for** $x_j \in \mathcal{N}(x_i)$ **do**
3: **for** $\forall d_j \in \mathcal{D}_j$ /*For each value of x_j*/ **do**
4: /*Assess coordination relation for neighbour variable*/
5: $\Delta_j^i(d_j) = \Delta_j^i(d_j) + \rho \cdot \left(\eta_{ji}.\mathcal{U}_j^j(d_j) - \mathcal{U}_j^i(d_j)\right)$
6: **end for**
7: **for** $\forall d_i \in \mathcal{D}_i$ /*For each value of x_i*/ **do**
8: /*Assess a coordination (balancing) relation for its own variable*/
9: $\bigtriangledown_i^i(d_i) = \bigtriangledown_i^i(d_i) + \rho \cdot \left(\eta_{ji}.\mathcal{U}_i^j(d_i) - \mathcal{U}_i^i(d_i)\right)$
10: **end for**
11: **end for**
12: **return** $\{\Delta\}_i$;

On the one hand, each agent a_i updates for each of the variables of its neighbours $x_j \in N(x_i)$ the value of the coordination relation (Δ_j^i) that quantifies how much agent a_i must change its utility for x_j to agree with a_j's max-marginal. For instance, in figure 5.6(b), a_1 will modify its utility for x_2 to agree with a_2's max-marginal. An agent a_i assesses Δ_j^i as as the difference between the value of the a_j' max-marginal and the value of a_i local max-marginal, damped by parameter ρ (lines 3-6). In figure 5.6(b), a_1 obtains $\Delta_2^1(0) = 0$ and $\Delta_2^1(1) = -15$. These values indicate that whereas the utility for a_1's assignment ($x_2 = 1$) should be decreased, the utility for a_2's assignment ($x_2 = 0$) should remain the same because both a_1 and a_2 receive the same utility to set $x_2 = 0$. Notice that applying such utility changes would bring a_1's utilities closer to a_2's for variable x_2.

On the other hand, each agent a_i assesses the coordination relation for its own variable x_i to counterbalance the utility updates of its neighbours (lines 7-10). In figure 5.6(b), agent a_1 must counterbalance the updates that its neighbours (a_2 and a_3) make to approach a_1's assignment through their coordination relations (Δ_1^2 and Δ_1^3).

Then, each agent uses its coordination relations to update its local subproblem in the `updateSubproblem` function (line 6, algorithm 5). In figure 5.6(b), a_1 updates its

subproblem by adding $\bigtriangledown_1^1$ for variable x_1, $\bigtriangleup_2^1$ for variables x_2, and $\bigtriangleup_3^1$ for variable x_3. Thereafter, each agent a_i solves its new subproblem to obtain its optimal local solution, $x^{*,i}$, along with its value $R(x^{*,i})$. Figure 5.6(b) shows the local solutions and its values computed by agents when setting $\rho = 0.5$. Likewise DaCSA, agents a_1 and a_3 change their assignments with respect to the first iteration: agent a_1 sets its variable x_1 to 1 and variables x_2, x_3 to 0. Likewise agent a_3 sets its variable x_3 to 1 and variable x_1 to 0. Thus, as a result agents get closer to an agreement because now agent a_1 agrees on the assignments of the shared variables with agent a_2. Moreover, all agents obtain lower utilities for their new assignments (compare the values of $R^1(x^{*,1}), R^2(x^{*,2})$ and $R^3(x^{*,3})$ with those in figure 5.2(a)). The sum of the utilities of subproblems solutions for this division, $5 + 5 + 6.5 = 16.5$, is lower than that obtained for the initial division and those obtained for DaCSA in section 5.3. Since the sum of utilities of subproblems' solutions is an upper bound on the quality of the optimal solution, agents obtain a tighter bound than those obtained in DaCSA after the first iteration. At the end of each iteration, agents check if some termination condition is satisfied (line 18, algorithm 5). In addition to the termination conditions listed for the generic DaC algorithm, EU-DaC also terminates when the max-marginal utilities are equal across agents because their subproblems will not change when updating after that point. Thus, unlike DaCSA, agents running EU-DaC can detect convergence even when they have not found the optimal solution.

5.4.3 Complexity analysis

We can directly assess the complexity of EU-DaC from the complexity of the generic DaC algorithm and from the particular EU-DaC implementations of the *wrapCoordinationInfo* and *updateCoordinationRelations* functions. EU-DaC implements the *wrapCoordinationInfo* function by assessing a message from a_i to a_j that contains the max-marginals for each of their shared variables. Therefore, the size of the coordination messages is linear to the domain of variables. Moreover, in EU-DaC each agent updates the coordination parameters requiring a number of operations linear in the size of the local relations. As a result, EU-DaC is a low-overhead algorithm because agents exchange a linear number of messages of linear size and performs a linear number of operations.

5.5 Empirical evaluation

In this section we provide an empirical evaluation of the two DaC algorithms proposed in this chapter: DaCSA and EU-DaC. We also provide an illustration of the tightness of the per-instance quality guarantees of DaC algorithms on DCOP problems. Firstly, we explain the details of our experimental setup in section 5.5.1. Secondly, we analyze our empirical results in section 6.3.4.

5.5.1 Empirical settings

Problem generation

We perform our comparison over randomly generated DCOP problems with binary variables. The process of generating a DCOP is divided in two steps. Firstly, we generate a constraint graph, and afterwards we generate values for each of the relationships in the constraint graph. Several results in agent research have found that the network topology has a significant effect when solving a distributed problem (e.g. emergence of social conventions (Pujol et al., 2005) or organizational adaptation (Gaston and DesJardins, 2005)). In our experiments we analyze three network topology alternatives:

Small-world. Many real-world networks, such a as food chains, electric power grids or social influence networks show the *small-world effect* (Mark, 2003), that is, the distance between any two nodes in the network is very small. We generate constraint graphs that show the small-world effect using the model proposed in (Newman and Watts, 1999). The graphs are created by starting from a ring and adding a small number of random edges. In particular, for each node we use a probability $p = 0.3$ of adding a new random edge.

Regular grids. The constraint graphs are rectangular grid where each agent is connected to its four closer neighbors.

Random networks. The constraint graphs are created by randomly adding three links for each variable.

Once a constraint graph is generated, we must assess its constraints' values. We are interested in evaluating our algorithms in the presence of strong dependencies among the values of variables. At this aim, we generate constraint values by following an Ising model (Baxter, 1982). Ising models have been widely used in statistical physics. Following an Ising model, the weight of each binary relation r_{ij}, is determined by first sampling a value κ_{ij} from a uniform distribution $U[-\beta, \beta]$ and then assigning

$$r_{ij}(x_i, x_j) = \begin{cases} \kappa_{ij} & x_i = x_j \\ -\kappa_{ij} & x_i \neq x_j \end{cases}$$

Note that the constraint pushes both variables to be similar when κ_{ij} is positive and forces them to be different when κ_{ij} is negative. The β parameter controls the average strength of interactions. In our experiments we set β to 1.6. The weight for each unary constraint r_i is determined by sampling κ_i from a uniform distribution $U[-0.05, 0.05]$ and then assigning $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$.

Algorithm's parameters

In what follows we provide details on the particular parameters selected for the each of the benchmarked algorithms in these experiments:

DaC algorithms. For each DaC algorithm we must specify the strategy used by agents to generate candidate solutions at each pair of *divide* and *coordinate* stages. We use the two strategies proposed in section 5.2. At each iteration, each agent generates two candidate solutions for its variable: (1) the assignment in which more agents agree on (S1); and (2) the assignment in which more agents agree on when the remaining variables in its subproblem are given by the assignments selected by the candidate solution in the previous iteration (S2). Then we set the parameters specific for each DaC algorithm as follows:

DaCSA. For DaCSA we use the same step-size at each iteration t as follows:

$$\epsilon = \frac{1+m}{t+m} \cdot \frac{R(x^{DaC}) - ub}{(\sum_{\lambda \in \{\lambda\}} (\lambda^t - \lambda^{t-1})^2)^2} \tag{5.18}$$

where $m = 5$, ub is the lowest upper bound calculated by agents up to that execution point and $R(x^{DaC})$ is the value of the DaCSA anytime solution. The intuition behind this formula is that the information transferred between agents for each constraint (ϵ) gets larger when the distance between the value of the best solution found so far and the bound grows, namely, when the algorithm is far from the optimal solution. Furthermore, it also gets larger when the level of disagreement among agents is smaller (there are fewer constraints among which we have to share the load). Each agent instantiates the bound, the value of the anytime configuration and the sum of Lagrange multipliers in equation 5.18 with the values of the last known *divide* and *coordinate* stages. At an early stage in the execution, when agents do not know yet the value of these parameters, they use a constant step-size $\epsilon = 0.001/\sqrt{t}$.

EU-DaC. For EU-DaC we set the value of the damping parameter ρ to 0.5.

DSA. For DSA, reviewed in chapter 2, we use an activation probability $p = 0.7$, a value that is reported to work well in (Zhang et al., 2005). Since DSA usually converges in a small number of iterations to get a fair comparison we restart it every time it converges keeping the best configuration among all converged solutions.

MGM. For MGM-$\{2,3\}$[3], reviewed in chapter 2, we set the probability q of being an offerer to 0.9, a value that is shown to reach the highest average solution quality by the experiments reported in (Maheswaran et al., 2004a).

Measures

We compare these algorithms based on the solution obtained in a number of message cycles. The number of message cycles is a commonly used measure for algorithm efficiency in the DCOP literature (Pearce and Tambe, 2007; Modi et al., 2005; Mailler and Lesser, 2004). It is specially adequate to our case because all the algorithms benchmarked are low-overhead algorithms.

[3]For MGM-$\{2,3\}$ we use the code provided in *http://teamcore.usc.edu/dcop/*

To normalize plots, we compare algorithms based on the percent gain. We assess the percent gain of an algorithm A with respect to an algorithm B at iteration t as $100 \cdot (\frac{q_A - q_B}{q_B})$, where q_A is the value of the solution of A algorithm and q_B is the value of the solution of B algorithm. Notice that positive values of the percent gain of an algorithm A with respect to an algorithm B stand for positive gains of algorithm A (the higher stand for better).

5.5.2 Results

In this section, we benchmark DaC algorithms against other state-of-the-art DCOP incomplete algorithms in terms of its solution quality and the quality of their bounds. First, we benchmark DaCSA against Max-sum (Farinelli et al., 2008) and DSA (Zhang et al., 2005), two state-of-the-art DCOP algorithms, that as reviewed in Chapter 3, can not provide any quality guarantee over their solutions. Second, we compare the quality of the solutions and the tightness of the bounds of EU-DaC and DaCSA with MGM algorithms (Maheswaran et al., 2004a; Katagishi and Pearce, 2007), that as reviewed in Chapter 3, are algorithms that provide system designer's quality guarantees.

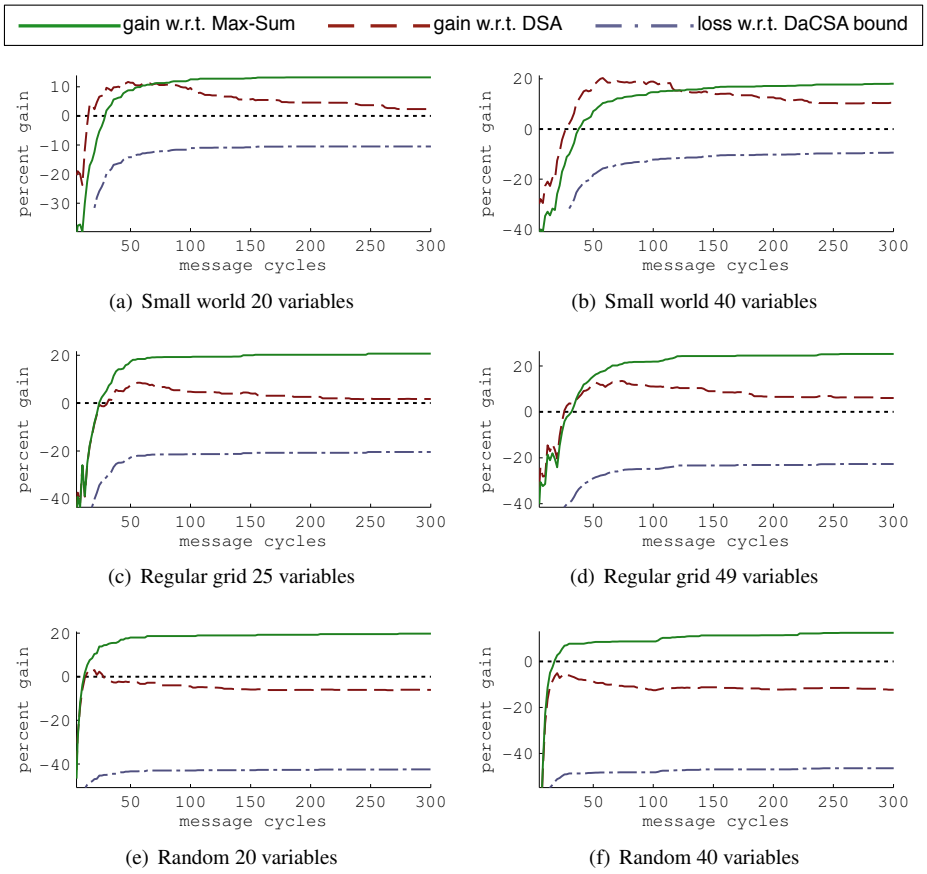
Comparing DaCSA with DSA and Max-Sum

In this section we analyse the results obtained when benchmarking DaCSA against the Max-sum and DSA algorithms.

As to the solution quality, figures 5.7 (a) (c) and (e) show the results for 20 agents (25 in the regular grid) on a small-world, regular grid, and random structure respectively. Each graph shows the mean among 25 problem instances of the percent gain of DaCSA with respect to Max-sum (MS) and DSA when varying the number of message cycles (mcs) up to 300. We also plotted the percent loss of DaCSA against the bound to have an idea of the accuracy of the bound. Over more realistic topologies (small world and regular grids) we observe that DaCSA outperforms Max-Sum and DSA. Concretely, at 50 message cycles, in small-world topologies, DaCSA gets a mean percent gain of around 10% with respect DSA and Max-sum. For regular grids, at 50 message cycles, the mean percent gain of DaCSA with respect to DSA and Max-sum is around 10% and 20% respectively. In both topologies, the gain with respect Max-sum remains nearly unchanged but the gain with respect to DSA is reduced although it never gets negative along the 300 messages cycles. However, in random instances DaCSA performs slightly worse and, although it gets better results than Max-sum (it gets a mean gain of 20% at 50 message cycles) it is unable to outperform DSA. The same conclusions (although more significant) hold for figures 5.7 (b), (d), and (f), as the number of variables increases to 40 (49 in the regular grid case). Hence, the network topology seems to be a key factor for DaCSA's performance.

Regarding the quality of the bound returned by DaCSA, table 5.3 shows the quality guarantees provided by the DaCSA by showing the mean of the approximation ratios given by DaCSA solution and its bound and its variance on different topologies.

The bound provided by DaCSA on realistic topologies is very accurate. The mean loss of DaCSA with respect to the bound is around 12% (approximation ratio of 1.17) in



(a) Small world 20 variables

(b) Small world 40 variables

(c) Regular grid 25 variables

(d) Regular grid 49 variables

(e) Random 20 variables

(f) Random 40 variables

Figure 5.7: Graphs showing the percent gain of DaCSA with respect to MS and DSA and the percent loss with respect to the DaCSA bound vs the number of message cycles on agent networks with different topologies and scales.

Topology	**Vars**	**Message cycles**		
		50 mcs	**100 mcs**	**300 mcs**
Small-world	v20	1.17 ± 0.004	1.13 ± 0.002	1.12 ± 0.002
	v40	1.22 ± 0.004	1.14 ± 0.002	1.10 ± 0.002
Regular grids	v25	1.32 ± 0.006	1.28 ± 0.006	1.26 ± 0.004
	v49	1.41 ± 0.004	1.33 ± 0.003	1.29 ± 0.003
Random	v20	1.77 ± 0.007	1.76 ± 0.007	1.75 ± 0.007
	v40	1.96 ± 0.007	1.95 ± 0.012	1.88 ± 0.008

Table 5.3: Mean of approximation ratios on different topologies calculated from DaCSA solutions and bounds

small-world and around 23% (approximation ratio of 1.32) in regular grids. The bound provided by DaCSA over random networks is less accurate.

These empirical results show the potential of DaCSA because it provides good solutions on realistic topologies. Moreover, the results also show that the approximation ratio that the algorithm provides in this kind of problems is significant. However, DaCSA provides worse solutions on randomly structured problems.

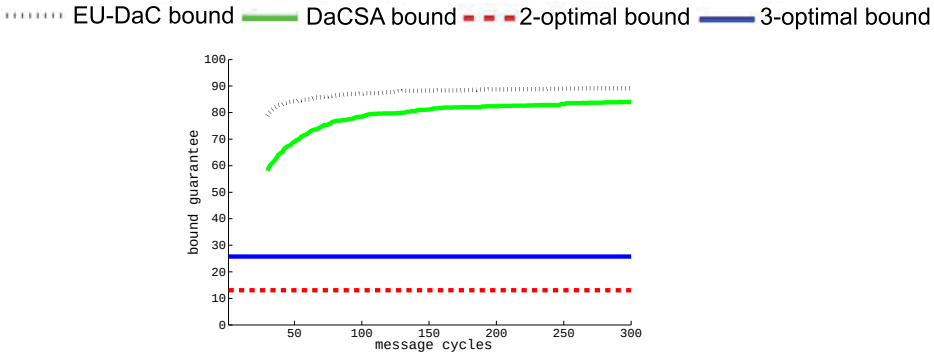
Comparing EU-DaC with DaCSA and MGM algorithms

In this section we analyse the results obtained when benchmarking EU-DaC and DaCSA against MGM-$\{2,3\}$ algorithms. Next, some comments over the quality guarantees provided by each algorithm are in place. Recall that, although both DaC algorithms and MGM algorithms can provide quality guarantees, following the approximate quality guarantees classification introduced in Chapter 1, these quality guarantees are of different nature. On the one hand, the quality guarantees of DaC algorithm (EU-DaC and DaCSA) are *per-instance* that can used from an agent perspective: agents assesses a the percent bound over the best candidate solution valuated so far. On the other hand, MGM quality guarantees are defined as per-structure class or problem independent guarantees that can be used from a system designer perspective and that are definer over the k-optimal solutions that MGM achieve on convergence. In our experiments we plot the tight k-optimal *graph-based quality guarantees* (Pearce and Tambe, 2007) that use knowledge of the problem graph structure and for which agents need to solve linear problem to assess them.

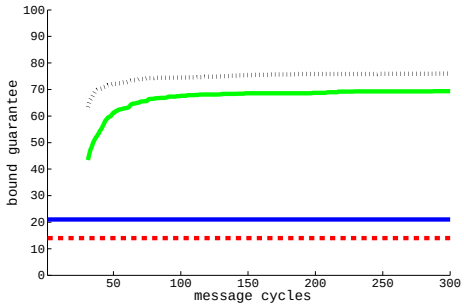
Regarding solution quality, figures 5.9 (a), (b) and (c) show the results for 100 agent networks on small-world, regular grid and random topologies respectively. Each graph shows the mean over 25 instances of the percent gain of EU-DaC respect to DaCSA and MGM-$\{2,3\}$. First, observe that in all experimented topologies EU-DaC outperforms DaCSA, the other DaC algorithm. Thus, results show that when agents explicitly communicate their max-marginal utilities instead of their local solutions they obtain results closer to an agreement. Observe that while EU-DaC obtains higher gains (around 40-60%) on structured topologies (small-world and regular grids), as the number of message cycles increases, these gains reduce to around 10%. In contrast, as to random networks, the gains of EU-DaC with respect to DaCSA are initially lower (around 30%) but remain more constant as the number of message cycles increases.

Secondly, when comparing with MGM algorithms, we observe that EU-DaC outperforms MGM-2 in all the scenarios and the same applies to MGM-3 on structured topologies. The gains of EU-DaC with respect to MGM-$\{2,3\}$ are lower than respect to DaCSA (around $5 - 10\%$). As to random topologies, EU-DaC even obtains sligth losses with respect to MGM-3 in the long run. Thus, we can conclude that EU-DaC is very competitive when compared with MGM-$\{2,3\}$ getting similar results and even outperforming them on some problem topologies.

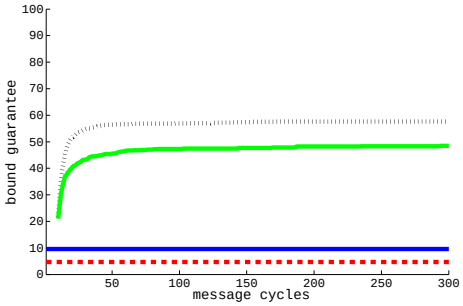
Regarding bound quality, in figure 5.8 we compare the quality guarantees of EU-DaC with those of DaCSA and MGM-$\{2,3\}$ by plotting the percent bound quality of their solutions when varying the number of message cycles. We assess the percent bound quality of an algorithm A as $100 \cdot \frac{q_A}{ub_A}$ where q_A is the value of the solution of A algorithm and ub_A is an upper bound on the value of the optimal solution. Intuitively, a



(a) Small world 100 variables



(b) Regular grids 100 variables



(c) Random 100 variables

Figure 5.8: Percent bound qualities of EU-DaC, DaCSA and k=$\{2,3\}$ optimal over different topologies

percent bound quality P indicates that the anytime solution has at least a P percent of the quality of the optimal. Figure 5.8 shows the mean of percent bound qualities provided by DaC algorithms (EU-DaC and DaCSA) and MGM-$\{2,3\}$ graph-based guarantees of over their converged after convergence on the different topologies. Firstly, we observe that DaC bounds are significantly more accurate than k-optimal bounds. While

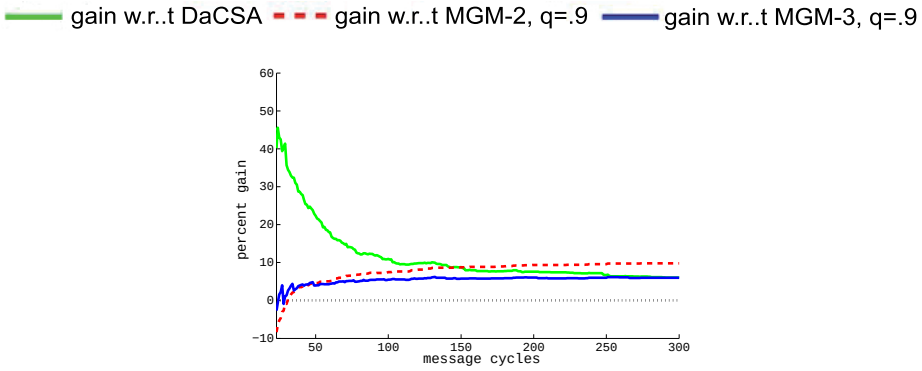
Topology	%Conv.	Conv. mcs
MGM-2		
Small-World	100%	36.8 ± 22.4
Regular grid	100%	44.8 ± 24.7
Random	100%	74.8 ± 64.6
MGM-3		
Small-World	80%	139.6 ± 74.7
Regular grid	40%	192.5 ± 75.3
Random	40%	181.3 ± 79.3

Table 5.4: Percentage of instances in which MGM algorithms converged in less than 300 mcs (%Conv.) and the mean number of mcs required (Conv. mcs).

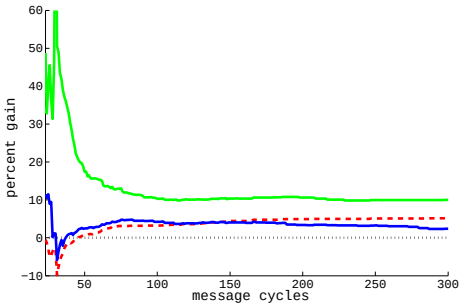
EU-DaC gets bounds around 55-85% (depending on the topology), 2- and 3-optimal bounds never go above 15% and 30% respectively in any scenario. Secondly, results show bounds provided by EU-DaC are always higher (around 5-10%) than those provided by DaCSA. It is not surprising since DaC algorithms use the quality of the best solution, which is higher for EU-DaC, to assess the bound.

In addition to its accuracy, it is also important when agents can start providing quality guarantees over their solutions. DaC agents require an initial number of iterations to distributedly assess the bound. These initial delays are illustrated in figure 5.8 when plotting DaC quality guarantees. In contrast, MGM algorithms can only provide quality guarantees on convergence. Figure 5.8 (d) shows a table with the percentage of instances in which MGM algorithms converged in less than 300 mcs and the mean number of mcs required. Observe that MGM-2 converges in all instances although the number of message cycles required to converge vary depending on the problem and the topology. As to MGM-3, however, the percentage of converged instances is quite low, particularly for regular grids and random topologies: MGM-3 fails to converge in more than half of the instances. Thus, for many instances, MGM-3 can not provide the 3-optimal bounds plotted in figure 5.8. Moreover, convergence requires a large number of message cycles.

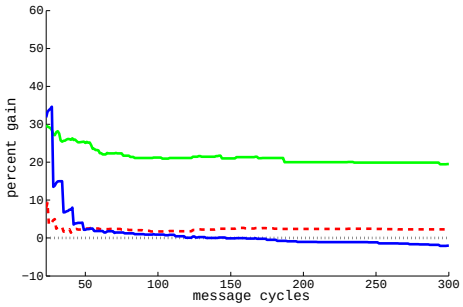
In summary, results in this section show how EU-DaC outperforms DaCSA in all experimented scenarios, confirming the intuition that coordinating by exchanging utilities for assignments instead of preferred assignments leads to solutions of higher quality. Regarding k-optimal MGM algorithms, results also show that EU-DaC solution quality is similar to MGM-$\{2,3\}$ algorithms. However, as to the bound quality, EU-DaC bounds, and in general DaC bounds, are significantly more accurate than 2- and 3- size optimal bounds, and, therefore, more meaningful for agents at run time. These results are on the line of what is discussed in chapter 1 that the more specific the quality guarantees the tighter they are likely to be. Moreover, they support the need, argued also in Chapter 1 for a broader concept of approximate quality guarantees that satisfy the different requirements derived from different perspectives, namely from the agents (as DaC quality guarantees) and from the system designer (as k-optimal guarantees).



(a) Small world 100 variables



(b) Regular grids 100 variables



(c) Random 100 variables

Figure 5.9: Percent gain of EU-DaC with respect to DaCSA, MGM-{2,3} over different topologies.

5.6 Conclusions

In this chapter we have contributed to overcome the limitations of incomplete DCOP algorithms on providing quality guarantees from an agent prespective as identified in chapter 3. Along this line, the main contribution of this chapter was the formalisation

of the DaC approach, a novel approach to solve DCOPs that provide agents' quality guarantees. Hence, according to DaC agents aim to solve DCOPs by searching for a division into subproblems such that individual subproblems agree on their solutions. DaC agents iteratively search through this space of divisions by: (i) exchanging information about their conflicts; (ii) updating their subproblems by exchanging utilities with their neighbours at the aim of reaching an agreement.

We explained how the DaC approach leads to different DaC algorithms depending on: (i) what information is exchanged about local subproblems; and (ii) which strategy agents follow to update them. Thus, the second contribution of this chapter is the formalisation of two different DaC algorithms: DaCSA and EU-DaC. Regarding the information exchanged, DaCSA agents exchange local solutions, whereas EU-DaC exchange max-marginals. Regarding the strategy used to update subproblems, a DaCSA agent exchanges some utility for its local solution when those differ, whereas an EU-DaC agent exchanges utilities to get closer max-marginals. Empirical results show that EU-DaC obtains better solutions than DaCSA. This supports the hypothesis that coordinating by exchanging utilities instead of optimal solutions leads to higher quality solutions.

Both DaC algorithms allow agents to provide anytime solutions with per-instance quality guarantees while using little local computation and local communication. Moreover, empirical results show how DaC algorithms leads (on average) to better solutions than other state-of-the-art DCOP algorithms that can not provide any quality guarantee over their solutions, namely DSA or Max-sum. Empirical results also show that when benchmarked against k-optimal MGM algorithms, EU-DaC solution quality is similar to MGM-$\{2,3\}$ algorithms, whereas EU-DaC quality guarantees, and in general DaC guarantees, are much tighter and, therefore, more meaningful for agents at run time. Hence, we can conclude that that the more the specificity of the quality guarantees, the tighter they are likely to be.

Figure 5.5 shows the resultant DCOP landscape after incorporating the aforementioned contributions of this chapter. Observe that now the landscape includes DaC as a novel approach to solve DCOPs. Moreover, the inclusion of the DaCSA and EU-DaC algorithms extends the DCOP incomplete algorithms that can provide guarantees from an agent perspective (so far uniquely occupied by the bounded Max-Sum algorithm).

This chapter defined a family of DCOP incomplete algorithms that provide approximate quality guarantees from an agent perspective. However, as discussed in chapter 1 there is also the need of algorithms that can provide approximate quality guarantees from a system designer perspective. Empirical results provided in this chapter show that k-size optimal MGM algorithms are competitive with EU-DaC in terms of solution quality whereas providing quality guarantees, complementary to DaC guarantees, that satisfy the requirements for the system designer. Therefore, in the next chapter we focus on generalising these k-size optimal quality guarantees, together with the algorithms that return k-size optimal solutions, focusing on DCOP incomplete algorithms that provide system designer' quality guarantees.

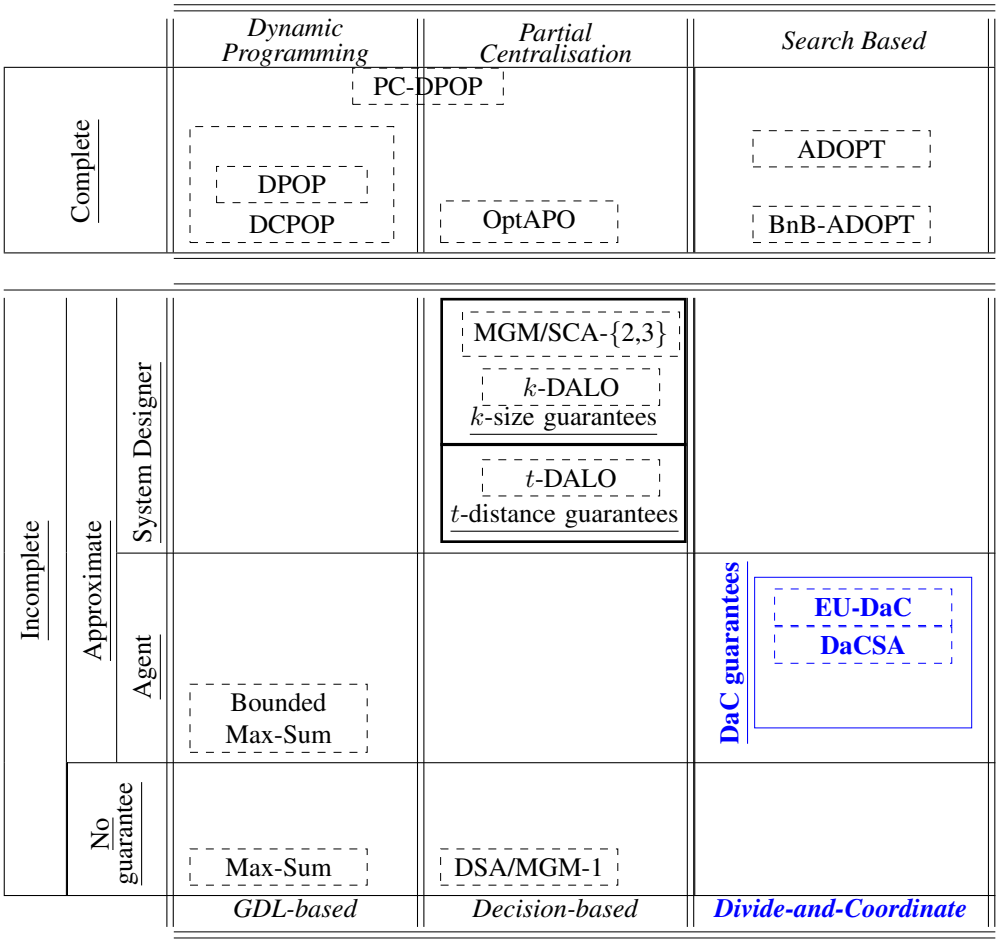


Table 5.5: DCOP algorithms landscape after Divide-and-Coordinate. Contributions of this chapter are highlighted in blue/bold. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

Chapter 6

Region Optimality

The family of DaC algorithms proposed in chapter 5 provides quality guarantees that are suitable to be used by agents, at runtime. In contrast, in this chapter we focus on incomplete DCOP algorithms that provide approximate system designer's quality guarantees.

To achieve that purpose we start from the k-size and t-distance optimal frameworks (Pearce and Tambe, 2007; Kiekintveld et al., 2010), the only DCOP frameworks that provide quality guarantees such that fulfill the system designer's requirements. Thus, k-size and t-distance define quality guarantees over their solutions at design time that apply: (i) to any problem (problem-independent guarantees); or (ii) to problems with a particular graph structure (per-structure guarantees). Size and distance optimality differ on the criteria to define local optimality. On the one hand, k-size optimality defines guarantees for solutions that cannot be improved by any group of k or fewer agents changing their decision. On the other hand, t-distance optimality defines guarantees for solutions that can not be improved by any group of agents distance at most t from a central agent changing their decision.

Although k-size and t-distance are the criteria explored so far in the literature, it is reasonable to wonder whether there are further local optimality criteria that can lead to better solution qualities while providing quality guarantees. In this chapter we set the foundations to explore this fundamental research question. First of all, we generalise the k-size and t-distance optimal frameworks to introduce region optimality, a flexible framework that provides quality guarantees for local optima in regions characterised by any arbitrary criterion. To achieve that purpose, we show how to compute problem-independent and per-structure guarantees for any region optima. Finally, we show how to extend these region optimal quality guarantees to exploit a-priori knowledge of the reward structure along the lines of the work in (Bowring et al., 2008) for k-size optima.

As a second contribution, this chapter shows how region optimality allows us to explore the space of local optimality criteria (beyond size and distance) looking for those criteria that lead to better solution qualities. First, we propose a novel criterion to define regions, the so-called size-bounded-distance criterion, which we design to overcome the main drawbacks of size and distance optimality. Secondly, we extend the DALO algorithm proposed in (Kiekintveld et al., 2010) for k-size and t-distance optimal solutions

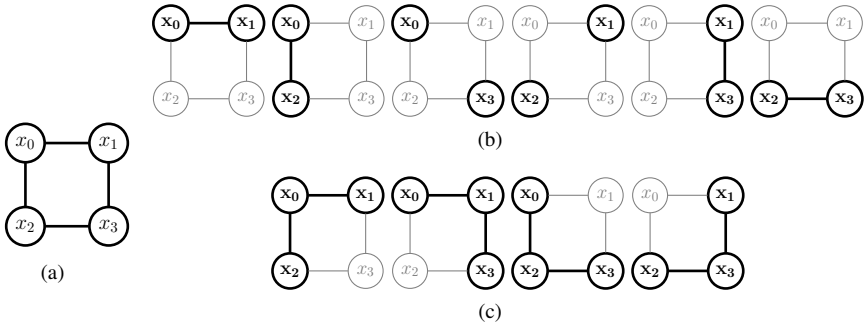


Figure 6.1: Example of (a) a DCOP graph, (b) its 2-size region and (c) its 3-size region.

to compute region optimal solutions. Finally, we benchmark the average performance of the generic region optimal algorithm when using the different region optima criteria proposed.

This chapter is organised as follows. Section 6.1 provides some background on the k-size and t-distance optimal frameworks. Section 6.2 introduces the notion of region optimal solution and the mechanisms for computing problem-independent and per-structure region optimal quality guarantees. Moreover, it also proves that the region optimality framework generalises both k-size and t-distance optimality. Section 6.3 introduces a new local optimality criterion, the so-called *size-bounded distance* criterion, and empirically compares it with respect to k-size and t-distance. Section 6.4 extends region optimality guarantees to exploit a-priori knowledge of the reward structure of the problem, if available. Finally, Section 6.5 draws conclusions and summarises the contributions of this chapter.

6.1 Background: size and distance optimality

As discussed in chapter 3, an important approach to incomplete DCOP algorithms focuses on coordinating the decisions of local groups of agents, instead of having each agent make an individual choice. So far two important local optimality criteria that establish how to group agents to coordinate their decisions have been explored: k-size (Pearce and Tambe, 2007) and t-distance (Kiekintveld et al., 2010) optimality.

An assignment x^k is a k-size optimum when no group of k or fewer agents can improve its reward $R(x^k)$ by simultaneously changing their variable assignments. On the other hand, t-distance optimality defines locality based on a group of surrounding nodes within a fixed distance t of a central node. For instance, figures 6.2(b) and 6.2(c) depict the groups of agents at distance 1 and 2 respectively for each agent in the DCOP in figure 6.2(a). Likewise k-size optimality, a t-distance optimum occurs when no group of t-distance agents can improve its reward.

A key property of k-size and t-distance optimality is that they are the only DCOP frameworks that can provide worst-case guarantees on the solution quality of k-size

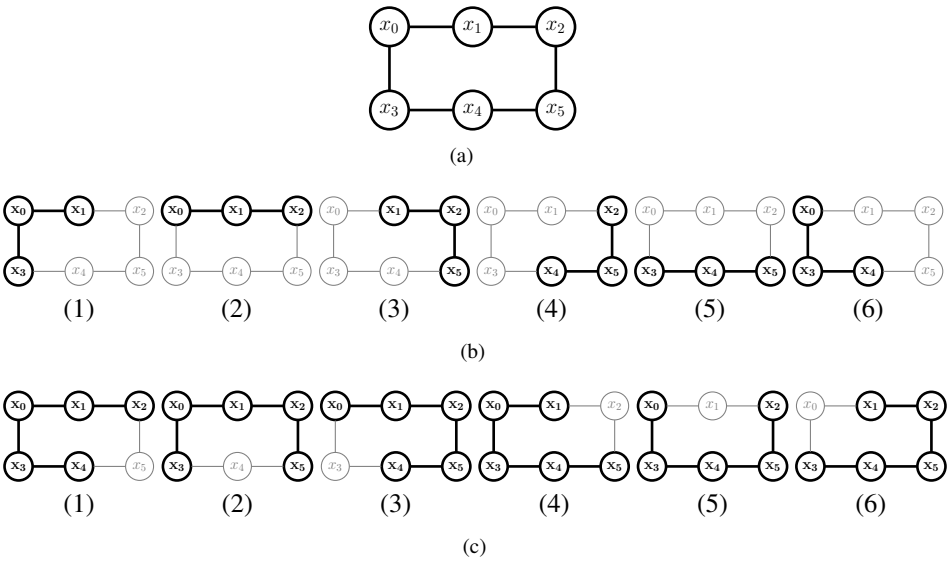


Figure 6.2: Example of (a) a DCOP graph, (b) its 1-distance region and (c) its 2-distance region.

and t-distance optimal solutions exploiting different levels of knowledge of the particular problem instance(s). Hence, k-size and t-distance optimal quality guarantees are problem-independent and per-class quality guarantees assessed at design time that can be used for the system designer to trade-off quality versus cost.

Both k-size and t-distance optimality have explored different mechanisms for computing bounds. Firstly, both k-size and t-distance optimality provide means for computing bounds independently of the problem instance (Pearce and Tambe, 2007; Kiekintveld et al., 2010), disregarding the graph structure and reward structure. Secondly, knowledge of a problem instance can be used to obtain tighter guarantees. One way is to exploit the knowledge about the graph structure of the DCOP (e.g. star, ring) (Pearce and Tambe, 2007) defining per-structure quality guarantees. Another way is to exploit the reward structure (Bowring et al., 2008) defining per-reward quality guarantees. We can group such mechanisms based on their computational costs.

On the one hand, a tight bound on the quality of every k-size or t-distance optimum can be computed using a linear program (LP) (Pearce and Tambe, 2007; Kiekintveld et al., 2010). In this method, rewards on the relations in the DCOP are treated as variables in a program whose goal is to minimise the quality guarantee. When the program is solved, the decision variables are instantiated with the values that, if used as relation rewards, would produce the DCOP whose local optimum has the lowest reward with respect to the global optimal solution. For example, for k-size optimality and for a specific graph structure, after running the program we obtain: (i) a quality guarantee δ for any k-size optimal solution on any DCOP having the specific constraint graph; and (ii) a DCOP having the specific constraint graph and a k-size optimal solution x^k whose

quality is equal to the relative error bound $0 \geq \delta \geq 1$, namely $R(x^k) = \delta \cdot R(x^*)$

On the other hand, there are methods that are computationally cheaper and can compute bounds in constant time (Pearce and Tambe, 2007; Kiekintveld et al., 2010). Despite the computational savings of these methods, with respect to the LP-based approach, in general tightness is not guaranteed.

Since k-size and t-distance optimal frameworks are algorithmic-independent, different algorithms have been proposed in order to efficiently search for k-size and t-distance optimal solutions. MGM-$\{2,3\}$ algorithms (Maheswaran et al., 2004a; Katagishi and Pearce, 2007) were the first k-size optimal algorithms that return 2-size and 3-size solutions respectively. Then, the work in (Katagishi and Pearce, 2007) formulated KOPT, a synchronous k-optimal algorithm that works for arbitrary settings of k. At the time of writing, the leading k-size/t-distance optimal algorithm is DALO (Kiekintveld et al., 2010), an asynchronous algorithm proposed to overcome the inefficiencies of KOPT that can find either k-size or t-distance optimal solutions for arbitrary settings of k and t. All these k-size/t-distance optimal algorithms proposed so far are decision-based: agents inside a neighbourhood coordinate to locally optimise their joint decision by considering any joint assignment that can improve their joint reward.

6.2 Generalizing size and distance optimality

In this section we generalize the concept of size and distance optimality to region optimality, which allows us to characterize any local optimum in a region $\mathcal{C}$ characterized by an arbitrary criterion. But before that, we analyse the commonalities between size and distance optimality.

The difference between k-size and t-distance optimal algorithms is the criterion employed to generate groups, that we shall refer to as *neighbourhoods*: k-size optimality creates neighbourhoods of a fixed size (k), whereas t-distance optimality creates per each agent a neighbourhood that includes all other agents within a certain distance (t) in the constraint graph. In both cases, we can regard a collection of neighbourhoods as an *exploration region*, namely $\mathcal{C}$, for either a k-size or t-distance optimal algorithm in a constraint graph. For instance, in figure 6.2(b), we show the neighbourhoods in the 1-distance region of the DCOP in figure 6.2(a), where boldfaced nodes in the constraint graph stand for variables included in the neighbourhood. Given some assignment x, we say that it is optimal in a neighbourhood $\mathcal{C}^\alpha \in \mathcal{C}$ if its reward cannot be improved by changing the values of some of the variables in the neighbourhood. For instance, the first graph on the left in figure 6.2(b) represents a neighbourhood composed of variables $\{x_0, x_1, x_3\}$. An assignment x is optimal in that neighbourhood if any other assignment that maintains the values of x_2, x_4 and x_5 receives at most the same reward as x. Then, we can claim optimality for x in a region $\mathcal{C}$ (noted as $x^{\mathcal{C}}$) whenever it is optimal in each neighbourhood in the region. For instance, an assignment x will be optimal in the region depicted in figure 6.2(c) if it is optimal in each of its neighbourhoods. Therefore, in general, for both k-size and t-distance based optimality, we observe that:

- each criterion is based on the definition of a region over the constraint graph; and

- given any assignment, checking for either k-size or t-distance optimality amounts to checking for optimality in that region.

Although k-size and t-distance are the criteria explored so far in the literature, it is reasonable to wonder whether there are further local optimality criteria that can lead to better solution qualities while providing quality guarantees.

Hereafter we propose a general notion of region optimality, the so-called $\mathcal{C}$-optimality, and describe how to calculate bounds for a $\mathcal{C}$-optimal assignment, namely an assignment that is optimal in an arbitrary region $\mathcal{C}$.

6.2.1 Region optimality

Next, we introduce the concepts of neighbourhood and region so that we can formally define region optimality. After that, we analyse the way in which neighbourhoods relate to each other by formalizing the idea that a larger neighbourhood *covers* a smaller one.

Formally, a neighbourhood is a subset of variables of $\mathcal{X}$. For instance, figure 6.2(b)(1) depicts a neighbourhood for the DCOP in figure 6.2(a) where bold-faced nodes in the constraint graph stand for variables included in the neighbourhood, namely $\{x_0, x_1, x_3\}$. Given two assignments x and y, we define $D(x, y)$ as the set containing the variables whose values in x and y differ. Given a neighbourhood A, we say that x is a *neighbour of y in A* iff x differs from y only in variables that are contained in A, thus $D(x, y) \subseteq A$. For example, consider that the variables in the DCOP of figure 6.2(a) take their values from $\{0, 1\}$. Then, given the solution $x = \{x_0 = 0, x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1\}$ and $y = \{x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1\}$, x is a neighbour of y in the neighbourhood of figure 6.2(b)(1) because $D(x, y) = \{x_0, x_1\} \subseteq \{x_0, x_1, x_3\}$.

Given some assignment x, we say that it is optimal in a neighbourhood A if its rewards cannot be improved by changing the values of some of the variables in the neighborhood. That is, for every assignment y such that x is a *neighbour of y in A*, we have that $R(x) \geq R(y)$. Thus, an assignment x is optimal in the neighbourhood of figure 6.2(b)(1) if any other assignment that maintains the values of x_2, x_4, x_5 as in x receives at most the same reward as x.

A region $\mathcal{C}$ is a multi-set[1] of subsets of $\mathcal{X}$, namely a multi-set of neighbourhoods of $\mathcal{X}$. For instance, figure shows a region composed of six neighbourhoods (1)-(6). Given a region $\mathcal{C}$, we say that x is *inside region $\mathcal{C}$ of y* iff there is a neighborhood $C^\alpha \in \mathcal{C}$ such that x is neighbour of y in C^α.

An assignment x is $\mathcal{C}$-optimal if it cannot be improved by any other assignment inside region $\mathcal{C}$ of x. That is, for every assignment y inside region $\mathcal{C}$ of x, we have that $R(x) \geq R(y)$. Thus, an assignment x in the region depicted in figure 6.2(b) if it is optimal in each of its six neighborhoods.

Relations among neighbourhoods

Given two neighbourhoods $A, B \subseteq \mathcal{X}$ we say that B completely covers A if $A \subseteq B$. We say that B does not cover A at all if $A \cap B = \emptyset$. Otherwise, we say that B covers

[1] A multi-set is a generalisation of a set that can hold multiple instances of the very same element.

A partially.

As an example of these relations, consider neighbourhoods (1) and (4) in figure 6.2(b), noted as $A = \{x_0, x_1, x_3\}$ and $B = \{x_2, x_4, x_5\}$ respectively, and neighbourhood (1) in figure 6.2(c), noted as $C = \{x_0, x_1, x_2, x_3, x_4\}$. Then, we have that A covers C partially (it contains some variables in C) whereas C covers A completely (C contains all variables in A). Moreover, A does not cover B at all and viceversa because these neighbourhoods do not have any variable in common.

Then, we say that $A \subseteq \mathcal{X}$ is covered by $\mathcal{C}$ if there is a neighbourhood $C^\alpha \in \mathcal{C}$ such that C^α completely covers A. For example, neighbourhood (1) in figure 6.2(b) is covered by the region of neighbourhoods in figure 6.2(c), because, among others, neighbourhood (1) in this region covers it completely.

For each neighbourhood C^α we can classify each relation S in a DCOP into one of three disjoint groups, depending on whether C^α covers S completely ($T(C^\alpha)$), partially ($P(C^\alpha)$), or not at all ($N(C^\alpha)$).

For each relation $r_V \in \mathcal{R}$ we define:

- $cc(r_V, \mathcal{C}) = |\{C^\alpha \in \mathcal{C} \text{ s.t. } V \subseteq C^\alpha\}|$, that is, the number of neighbourhoods in $\mathcal{C}$ that cover the domain of r_V completely. In the 2-distance region $\mathcal{C}$ in figure 6.2(c), $cc(r_{01}, \mathcal{C}) = 2$ because neighborhoods (b)(1) and (b)(2) completely cover $\{x_0, x_1\}$.

- $nc(r_V, \mathcal{C}) = |\{C^\alpha \in \mathcal{C} \text{ s.t. } V \cap C^\alpha = \emptyset\}|$, that is, the number of neighbourhoods in $\mathcal{C}$ that do not cover the domain of r_V at all. For example, in the 2-distance region $\mathcal{C}$ in figure 6.2(c), $nc(r_{01}, \mathcal{C}) = 2$ because neighborhoods (b)(4) and (b)(5) do not cover $\{x_0, x_1\}$ at all.

- $pc(r_V, \mathcal{C}) = |\{C^\alpha \in \mathcal{C} \text{ s.t. } V \subseteq C^\alpha \text{ and } V \cap C^\alpha \neq \emptyset|$ that is, the number of neighborhoods that partially cover the domain of r_V. Alternatively, the number of partial relations can also be defined in terms of the non-covered and the totally-covered relations as $pc(r_V, \mathcal{C}) = |\mathcal{C}| - nc(r_V, C) - cc(r_V, C)$. Thus, in the 2-distance region $\mathcal{C}$ in figure 6.2(c), $pc(r_{01}, \mathcal{C}) = 2$ because neighborhoods (b)(3) and (b)(6) partially cover $\{x_0, x_1\}$.

6.2.2 Fine quality guarantees for region optima

After its formal definition, we are interested in providing a relative error bound on the quality of any $\mathcal{C}$-optimal assignment in a DCOP with non-negative rewards. We say that we have a bound δ (being $0 < \delta \leq 1$) when we can state that the quality of any $\mathcal{C}$-optimal assignment $x^{\mathcal{C}}$ is larger than δ times the quality of the optimal assignment x^*. Hence, we are interested in providing a bound δ such that for every $x^{\mathcal{C}}$ we have that $\frac{R(x^{\mathcal{C}})}{R(x^*)} \geq \delta$. For a given set of relations $\mathcal{R}$, let $x^{\mathcal{C}}_-$ be the $\mathcal{C}$-optimal assignment with smallest reward, then $\frac{R(x^{\mathcal{C}}_-)}{R(x^*)}$ provides a tight bound on the quality of any $\mathcal{C}$-optimal assignment for the specific rewards $\mathcal{R}$.

We are interested in defining bounds that are independent of the particular reward values of the DCOP. In that setting, a simple way to provide a bound on the quality is to

directly search the space of reward values to find the set of rewards $\mathcal{R}^*$ that minimizes $\frac{R^*(x^{\mathcal{C}}_{-})}{R^*(x^*)}$. More formally, this can be encoded as:

Find $\mathcal{R}$, $x^{\mathcal{C}}$ and x^* that
minimize $\frac{R(x^{\mathcal{C}})}{R(x^*)}$
subject to $x^{\mathcal{C}}$ being a $\mathcal{C}$-optimal for $\mathcal{R}$

Based on the definition of region optimality in section 6.2.1, the condition of being $\mathcal{C}$-optimal can be expressed as: for each x inside region $\mathcal{C}$, $R(x^{\mathcal{C}}) \geq R(x)$. However, instead of considering all assignments inside region $\mathcal{C}$, given an assignment $x^{\mathcal{C}}$ we only consider the assignments such that the variables that deviate from $x^{\mathcal{C}}$ take the same value that they do in the optimal assignment x^*. If we restrict to this subset of assignments, then each neighbourhood covers $2^{|C^\alpha|}$ assignments, one for each subset of variables in the neighbourhood. Let 2^{C^α} stand for the set that contains all subsets of neighbourhood C^α. Then, for each $A^k \in 2^{C^\alpha}$ we can define an assignment x^{α_k} such that: (i) for every variable x_i in a relation completely covered by A^k we have that $x_i^{\alpha_k} = x_i^*$; and (ii) for every variable x_i that is not covered at all by A^k we have that $x_i^{\alpha_k} = x_i^{\mathcal{C}}$. Then, we can write the value of x^{α_k} as :

$$R(x^{\alpha_k}) = \sum_{r \in T(A^k)} r(x^*) + \sum_{r \in P(A^k)} r(x^{\alpha_k}) + \sum_{r \in N(A^k)} r(x^{\alpha_k}), \tag{6.1}$$

where $T(A^k)$ is the set of completely covered relations, $P(A^k)$ stands for the set of partially covered relations and $N(A^k)$ stands for the set of relations not covered at all. Now, we say that $x^{\mathcal{C}}$ is $\mathcal{C}$-optimal if it fulfils:

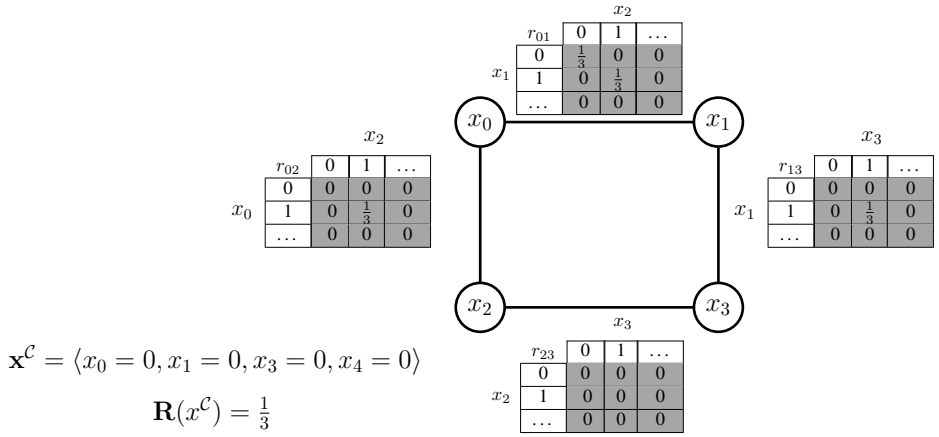
$$R(x^{\mathcal{C}}) \geq \sum_{r \in T(A^k)} r(x^*) + \sum_{r \in P(A^k)} r(x^{\alpha_k}) + \sum_{r \in N(A^k)} r(x^{\mathcal{C}}) \quad \forall A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\} \tag{6.2}$$

By setting partially covered relations to the minimum possible reward (that is to 0 assuming non-negative rewards), equation 6.2 results in:

$$R(x^{\mathcal{C}}) \geq \sum_{r \in T(A^k)} r(x^*) + \sum_{r \in N(A^k)} r(x^{\mathcal{C}}) \quad \forall A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\} \tag{6.3}$$

Given the definition of $\mathcal{C}$-optimality in equation 6.3, we can proceed on specifying the linear programming formulation of the initial problem. First, we assume without loss of generality, that $x^{\mathcal{C}}_{-} = \langle 0, \ldots, 0 \rangle$ and $x^* = \langle 1, \ldots, 1 \rangle$, where 0 and 1 stand for the first and second value in each variable domain. Second, we create two real positive variables for each relation $r \in \mathcal{R}$, one representing $r(x^{\mathcal{C}})$, noted as z_r, and another one representing $r(x^*)$, noted as y_r. Third, to obtain a linear program (LP) we can normalize the rewards of the optimal solution to add up to one ($\sum_{r \in \mathcal{R}} y_r = 1$) to turn it into a linear program. Fourth, we add all the constraints from equation 6.3 to guarantee the optimality of $x^{\mathcal{C}}$.

Applying these transformations, we can simplify the initial program into the following LP with z and y being vectors of positive real numbers:



$\mathbf{x}^{\mathcal{C}} = \langle x_0 = 0, x_1 = 0, x_3 = 0, x_4 = 0 \rangle$

$\mathbf{R}(x^{\mathcal{C}}) = \frac{1}{3}$

$\mathbf{x}^{*} = \langle x_0 = 1, x_1 = 1, x_3 = 1, x_4 = 1 \rangle$

$\mathbf{R}(x^{*}) = 1$

Figure 6.3: Example of a DCOP for which the fine 2-size region optimal bound $\delta = \frac{1}{3}$ that applies to the 2-size optimal solution $x^{\mathcal{C}}$ with respect to the optimal x^* is tight.

minimize $\sum_{r \in \mathcal{R}} z_r$
subject to
$\sum_{r \in \mathcal{R}} y_r = 1$
and for each neighbourhood $A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\}$ covered by $\mathcal{C}$ subject to
$\sum_{r \in \mathcal{R}} z_r \geq \sum_{r \in T(A^k)} y_r + \sum_{r \in N(A^k)} z_r$

where recall that $T(A^k)$ contains the relations completely covered by A^k, and $N(A^k)$ the relations that are not covered by A^k at all.

As an example, we turn back to figure 6.1 to assess the LP region optimal bound for the 2-size region in figure 6.1(b). In this case, we assume $x^{\mathcal{C}}_{-} = \langle x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0 \rangle$ and $x^* = \langle x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 1 \rangle$, where 0 and 1 stand for the first and second value in each variable domain. First, we create the real variables, two for each of the four relations. Thus, given the relation r_{01} we create two real variables: one representing the value of $x^{\mathcal{C}}_{-}$, $z_{r_{23}}$, and one representing the value of x^*, $y_{r_{01}}$. Finally, to guarantee the optimality of $x^{\mathcal{C}}$, we add six constraints, one for each neighbourhood that compose the 2-size region depicted in figure 6.1(b). Thus, for the neighborhood depicted on the left of figure 6.1(b) (composed of variables x_0, x_1), we impose via c0 that the value of $x^{\mathcal{C}}_{-}$ is greater than the sum of the values of totally covered relations for x^* ($y_{r_{01}}$) plus the values of non-covered relations for $x^{\mathcal{C}}_{-}$ ($z_{r_{23}}$). The resulting LP formulation is:

minimize $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}}$
subject to

$$y_{r_{01}} + y_{r_{13}} + y_{r_{23}} + y_{r_{02}} = 1$$

and subject to:

(c0) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{01}} + z_{r_{23}}$
(c1) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{02}} + z_{r_{13}}$
(c2) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq 0$
(c3) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq 0$
(c4) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{13}} + z_{r_{02}}$
(c5) $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{23}} + z_{r_{01}}$

After solving this LP, $\delta = \sum_{r \in \mathcal{R}} z_r$ is a tight bound on the quality of a $\mathcal{C}$-optimal solution for the graph structure represented by $\mathcal{R}$. Thus, by solving the LP for the 2-size optimal region in figure 6.1(b) we obtain bound $\delta = \frac{1}{3}$. Moreover, we can use the values of the instantiated real variables, corresponding to the relations' rewards for $x^{\mathcal{C}}_{-}$ and x^*, to generate DCOPs for which the assessed bound is tight. Figure 6.2.2 shows a DCOP with a reward structure for which the 2-size region optimal bound $\delta = \frac{1}{3}$ obtained for the constraint graph in figure 6.1(a) is tight. It is easy to see that value of the 2-size optimal $x^{\mathcal{C}}_{-} = \langle 0, 0, 0, 0 \rangle$ is $1/3$, higher than the value of any assignment inside the 2-size region, whereas the value of the optimal assignment $x^* = \langle 1, 1, 1, 1 \rangle$ is 1.

Let M be the number of variables of the largest neighbourhood in $\mathcal{C}$. The LP uses $2 \cdot |\mathcal{R}|$ variables and $\mathcal{O}(2^M \cdot |\mathcal{C}|)$ constraints, and hence it is solvable in time polynomial in $|\mathcal{R}|$ and in $2^M \cdot |\mathcal{C}|$.

6.2.3 Coarse quality guarantees for region optima

The computational complexity of the previous LP can be high as the number of relations $|\mathcal{R}|$, the number of neighbourhoods $|\mathcal{C}|$ or its size M grows. In this section we show that we can compute a bound in time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$. Furthermore, the result will prove as a very valuable tool for future theoretical developments. As a counterpart, we lose the tightness of the bound.

Proposition 6. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP with non-negative rewards and $\mathcal{C}$ a region. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then*

$$R(x^{\mathcal{C}}) \geq \frac{cc_*}{|\mathcal{C}| - nc_*} R(x^*) \tag{6.4}$$

where $cc_* = \min_{r \in \mathcal{R}} cc(r, \mathcal{C})$, $nc_* = \min_{r \in \mathcal{R}} nc(r, \mathcal{C})$, and x^* is the optimal assignment.

The proof for proposition 6 is a generalization of the one in Pearce and Tambe (2007) for k-optimality.

Proof. For every $C^\alpha \in \mathcal{C}$, consider an assignment x^α such that $x^\alpha_i = x^{\mathcal{C}}_i$ if $x_i \notin C^\alpha$ and $x^\alpha_i = x^*_i$ if $x_i \in C^\alpha$. Since $x^{\mathcal{C}}$ is $\mathcal{C}$-optimal, for all $C^\alpha \in \mathcal{C}$, $R(x^{\mathcal{C}}) \geq R(x^\alpha)$ holds, and hence

$$R(x^{\mathcal{C}}) \geq \frac{\sum_{C^\alpha \in \mathcal{C}} R(x^\alpha)}{|\mathcal{C}|}. \tag{6.5}$$

Now for each x^α, we have that $R(x^\alpha) = \sum_{r \in \mathcal{R}} r(x^\alpha)$.

We can split the sum into completely covered ($T(C^\alpha)$), partially covered ($P(C^\alpha)$), or not covered at all ($N(C^\alpha)$) relations, having $R(x^\alpha) = \sum_{r\in T(C^\alpha)} r(x^\alpha) + \sum_{r\in P(C^\alpha)} r(x^\alpha) + \sum_{r\in N(C^\alpha)} r(x^\alpha)$.

Then, by setting partially covered relations to the minimum possible reward (0 assuming non-negative rewards), $R(x^\alpha) \geq \sum_{r\in T(C^\alpha)} r(x^\alpha) + \sum_{r\in N(C^\alpha)} r(x^\alpha)$. Now, by definition of x^α, for every variable x_i in a relation completely covered by C^α we have that $x_i^\alpha = x_i^*$, and for every variable x_i in a relation not covered at all by C^α we have that $x_i^\alpha = x_i^\mathcal{C}$. Hence, $R(x^\alpha) \geq \sum_{r\in T(C^\alpha)} r(x^*) + \sum_{r\in N(C^\alpha)} r(x^\mathcal{C})$. To assess a bound, after substituting this inequality in equation 6.5, we have that

$$R(x^\mathcal{C}) \geq \frac{\sum\limits_{C^\alpha\in\mathcal{C}} \sum\limits_{r\in T(C^\alpha)} r(x^*) + \sum\limits_{C^\alpha\in\mathcal{C}} \sum\limits_{r\in N(C^\alpha)} r(x^\mathcal{C})}{|\mathcal{C}|}. \tag{6.6}$$

We need to express the numerator in terms of $R(x^\mathcal{C})$ and $R(x^*)$. Grouping the sum by relations and reminding that $cc_* = \min_{r\in\mathcal{R}} cc(r,\mathcal{C})$, the term on the left can be expressed as:

$$\sum_{C^\alpha\in\mathcal{C}} \sum_{r\in T(C^\alpha)} r(x^*) = \sum_{r\in\mathcal{R}} cc(r,\mathcal{C})\cdot r(x^*) \geq$$
$$\geq \sum_{r\in\mathcal{R}} cc_*\cdot r(x^*) = cc_* \sum_{r\in\mathcal{R}} r(x^*) = cc_*\cdot R(x^*).$$

Furthermore, recalling that $nc_* = \min_{r\in\mathcal{R}} nc(r,\mathcal{C})$, we can do the same with the right term:

$$\sum_{C^\alpha\in\mathcal{C}} \sum_{r\in N(C^\alpha)} r(x^\mathcal{C}) = \sum_{r\in\mathcal{R}} nc(r,\mathcal{C})\cdot r(x^\mathcal{C}) \geq$$
$$\geq \sum_{r\in\mathcal{R}} nc_*\cdot r(x^\mathcal{C}) = nc_* \sum_{r\in\mathcal{R}} r(x^\mathcal{C}) = nc_*\cdot R(x^\mathcal{C}).$$

After substituting these two results in equation 6.5 and rearranging terms, we obtain equation 6.4. □

Proposition 6 directly provides a simple algorithm to compute a bound. Given a region $\mathcal{C}$ and a graph structure, we can directly assess cc_* and nc_* by computing $cc(r,\mathcal{C})$ and $nc(r,\mathcal{C})$ for each relation $r \in \mathcal{R}$ and taking the minimum. This will take time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$, that is linear in the number of relations of the DCOP and linear in the number of neighbourhoods in the region.

As an example, now we turn back to figure 6.2 to assess the bounds for a $\mathcal{C}$-optimal assignment using equation 6.4. First, we assess the bound for the 1-distance region $\mathcal{C}_1$ in figure 6.2(b). Given the relation r_{01}, we assess the number of neighbourhoods that completely cover $\{x_0, x_1\}$ as $cc(r_{01},\mathcal{C}_1) = 2$ (the two first neighbourhoods on the left-hand side) and the number of neighbourhoods that do not cover $\{x_0, x_1\}$ at all as $nc(r_{01},\mathcal{C}_1) = 2$ (the fourth and fifth neighbourhoods). After repeating the process for the rest of relations in the constraint graph, we obtain that $cc_* = 2$ and $nc_* = 2$, and hence $\frac{cc_*}{|\mathcal{C}_1|-nc_*} = \frac{2}{6-2} = \frac{1}{2}$. Notice that this leads to a better bound that the one we

obtain following the result in (Kiekintveld et al., 2010), since $\frac{m+t-1}{n} = \frac{1}{3}$. This is due to the fact that we are computing the bound specifically for this graph structure, whilst the bounds provided in (Kiekintveld et al., 2010) are independent of the graph structure. If now we consider the 2-distance region $\mathcal{C}_2$ in figure 6.2(c), we obtain that $\frac{cc_*}{|\mathcal{C}_2|-nc_*} = \frac{4}{6-0} = \frac{2}{3}$. Again, this leads to a better bound than the one reported in (Kiekintveld et al., 2010) since $\frac{m+t-1}{n} = \frac{1}{2}$.

Note that the bounds provided as example are tight. However, despite of these examples, the bound assessed by proposition 6 is not guaranteed to be tight and can return worse bounds than the fine bounds computed by means of the LP. As example, consider the 2-size region $\mathcal{C}_2$ in figure 6.1(b) for which, in section 6.2.2, we assessed the $\mathcal{C}$-optimal as $\delta = \frac{1}{3}$. In this case, we obtain that $\frac{cc_*}{|\mathcal{C}_2|-nc_*} = \frac{1}{6-1} = \frac{1}{5}$ (each relation is totally covered and non-covered by one single neighborhood). Thus, this fine bound is not tight.

Both the LP and proposition 6 assess bounds that depend on the graph structure, the so-called per-structure class guarantees in chapter 1, but are independent of the specific reward values. We can always use them to assess bounds independently of the graph structure, assessing problem-independent guarantees, by computing the bound for the complete graph, since any other structure is a particular case of the complete graph with some rewards set to zero.

In the next two sections we show that the constant-time problem-independent bounds provided for size and distance optimality in (Kiekintveld et al., 2010; Pearce and Tambe, 2007) are particular cases of proposition 6.

6.2.4 Size-optimal bounds as a specific case of region optimal bounds

Now we present the main result in (Pearce and Tambe, 2007) as a specific case of region optimality. Recall that an assignment is k-size optimal if it can not be improved by changing the value of any group of size k or fewer variables.

Proposition 7. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP with non-negative rewards and m the maximum relation arity. Then, for any k-size optimal assignment x^k:*

$$R(x^k) \geq \frac{\binom{|\mathcal{X}|-m}{k-m}}{\binom{|\mathcal{X}|}{k} - \binom{|\mathcal{X}|-m}{k}} R(x^*) \tag{6.7}$$

Proof. This result is just a specific case of our general result where we take as region all subsets of size k, that is $\mathcal{C} = \{C^\alpha \subseteq \mathcal{X} \mid |C^\alpha| = k\}$. The number of neighbourhoods in the region is $|\mathcal{C}| = \binom{|\mathcal{X}|}{k}$. The number of neighbourhoods in $\mathcal{C}$ that completely cover r_V is $cc(r_V, \mathcal{C}) = \binom{|\mathcal{X}|-|V|}{k-|V|}$, where $|V|$ stands for the cardinality of V (take the variables in r plus $k - |V|$ variables out of the remaining $|\mathcal{X}| - |V|$). Because $cc(r_V, \mathcal{C})$ reaches the minimum value with the maximum value of $|V|$, $cc_* = \binom{|\mathcal{X}|-m}{k-m}$. The number of neighbourhoods in $\mathcal{C}$ that do not cover r_V at all is $nc(r_V, \mathcal{C}) = \binom{|\mathcal{X}|-|V|}{k}$ (take k variables out of the remaining $|\mathcal{X}| - |V|$ variables). Because $nc(r_V, \mathcal{C})$ reaches the minimum value with the maximum value of $|V|$, $nc_* = \binom{|\mathcal{X}|-m}{k}$. Finally, we obtain equation 6.7 by using $|\mathcal{X}|$, cc_* and nc_* in equation 6.4, and simplifying. □

6.2.5 Distance-optimal bounds as a specific case of region optimal bounds

Now we present the main result in (Kiekintveld et al., 2010) as a specific case of $\mathcal{C}$-optimality. First, let us notice that the bound in (Kiekintveld et al., 2010) can be more easily proved if the DCOP constraint graph is assumed to be connected. After that, we will see that the bound can be improved in the case that the DCOP constraint graph is composed of a set of connected components. Consider a connected DCOP with n variables, minimum constraint arity m, non-negative rewards, and globally optimal assignment x^*. It is easy to see that whenever $m + t - 1 > n$, the length of the shortest path between any two nodes is smaller than t, and hence any t-distance optimal assignment will in fact be globally optimal.

Proposition 8. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a connected DCOP with non-negative rewards. Then, whenever $m+t-1 \leq n$, we can bound the quality of any t-distance optimal assignment x^t as*

$$R(x^t) \geq \frac{(m+t-1)}{n} R(x^*) \tag{6.8}$$

Proof. This result is just a specific case of our general result where we take as region the t-distance neighbourhoods for each variable $x \in \mathcal{X}$, that is $\mathcal{C} = \{\Omega_t(x) |\ x \in \mathcal{X}\}$. The number of neighbourhoods in the region is $|\mathcal{C}| = n$. Next, we show that for every relation r, we have that the number of neighbourhoods in $\mathcal{C}$ that completely cover r, $cc(r, \mathcal{C})$ is at least $m + t - 1$. The only variables that do not have r in their t-distance neighbourhood are those variables that are at distance t or more from every variable in r. If no such variable exist, then $cc(r, \mathcal{C}) = n > m + t - 1$. Otherwise, let x' be one of these variables. There is a shortest path connecting x' to its closest variable in r (say x). The path must have length at least t, that is $x, x_1, \ldots, x_{t-1}, \ldots, x'$. Now, it is clear that r is in the t-distance neighbourhood of the $t-1$ variables $\{x_1, \ldots, x_{t-1}\}$. Note that since we are taking the shortest path to any variable in r, no x_i can be in r. Since r is also in the t-distance neighbourhood of every variable in r and there can be no intersection between r and $\{x_1, \ldots, x_{t-1}\}$, we have $cc(r, \mathcal{C}) = |r| + t - 1 \geq m + t - 1$. Hence $cc_* \geq m + t - 1$. By definition, $nc_* \geq 0$. Finally, we obtain equation 6.8 by using $|\mathcal{C}|$, cc_* and nc_* in equation 6.4, and simplifying. □

In case the DCOP is not connected, we can obtain a better bound by simply applying the bound previously stated for each connected component and taking the minimum. That is $R(x^t) \geq \frac{(m+t-1)}{n_*} R(x^*)$, where n_* is the number of elements of the largest connected component, which is always smaller than n.

6.3 Empirical Evaluation

In this section we show how we can benefit from the larger space of criteria for defining regions provided by region optimality. We start by analyzing the regions generated by k-size and t-distance on DCOPs with different structures, to conclude that k-size generates a potentially huge number of neighborhoods of limited size and t-distance generates a limited number of potentially huge neighborhoods. To keep under control

the amount and size of neighborhoods we introduce a new type of regions, namely size-bounded distance regions, which include a limited number of limited size neighborhoods. Finally, we empirically show that algorithms for approximate DCOP solving can benefit from using size-bounded distance regions.

We start by analyzing k-size and t-distance regions in section 6.3.1, to motivate the introduction of size-bounded distance regions in section 6.3.2. The DALO algorithm was proposed in (Kiekintveld et al., 2010) to find either k-size or t-distance optimal solutions. In section 6.3.3 we show how we can extend it to find an optimal in any region $\mathcal{C}$. Finally, in section 6.3.4 we compare the performance of size, distance and size-bounded distance regions on DCOPs with different graph structures using DALO.

6.3.1 Analysis of size and distance regions

We are interested in analyzing the regions generated by k-size and t-distance on DCOPs with different structures. More concretely, we want to assess the number of different neighbourhoods as well as the size (number of variables) for each neighbourhood, since both parameters strongly influence the amount of computation needed to obtain a k-size, t-distance optimum. The worst case time for checking optimality in a neighbourhood is exponential in its number of variables. Furthermore, if an agent has to consider a large number of neighbourhoods, it will have to share its time among them. Hence, in terms of computational effort, it is of interest to find regions that have a limited number of neighbourhoods of limited size.

According to k-size optimality, the size is limited by k but the number of neighbourhoods grows as $\binom{|X|}{k}$, which can turn out prohibitively large.

	Random		**Scale-free**		**NPLA**	
	MaxS	#	**MaxS**	#	**MaxS**	#
K5	5	167	5	963	5	11366
T1	10	1	27	1	63	1
T2	38	1	82	1	99	1
S5	5	3	5	3	5	10

Table 6.1: Statistics for regions generated by K5, T1, T2 and S5 criteria for 100 agents. MaxS stands for the maximum size of a neighbourhood and # for the average number of neighbourhoods per agent.

Following t-distance optimality, the number of neighbourhoods is $\mathcal{O}(|\mathcal{X}|)$, but the size of the neighbourhoods is not limited. For example, the 1-distance region of a complete graph contains a single neighbourhood with all the variables in the problem, and hence finding a 1-distance optimum in a complete graph is as hard as finding a global optimum.

For a more detailed empirical analysis, we have computed statistics of the maximum neighbourhood size in a region (MaxS) and the number of neighbourhoods per agent (#) over randomly generated constraint graphs. We have used three different types of graph structures: $G(n, M)$ random graphs (Bollobas, 2001), Barabasi-Albert (BA) scale-free graphs (Barabasi and Albert, 1999), and non-linear preferential attachment

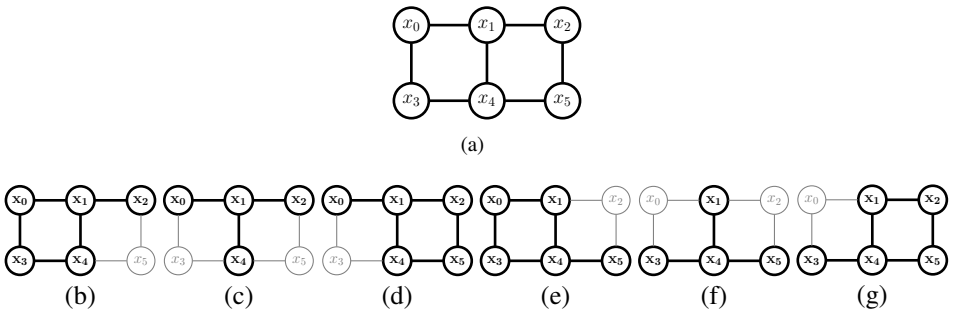


Figure 6.4: Example of (a) a DCOP graph, and (b)-(g) the set neighbourhoods for the 5-size-distance bounded region.

(NLPA) graphs based on the BA model, but with a larger emphasis on many nodes having fewer connections. All the graphs have 100 nodes with a density of four, meaning that on average each node has four neighbours. We compare the results of three different criteria: 5-size (K5)[2], 1-distance (T1) and 2-distance (T2). The first three rows in table 6.1 present the averages over 50 DCOPs of the maximum neighbourhood size in a region (MaxS) and the number of neighborhoods per agent (#) for each criteria and each type of graphs. From these statistics we observe that T1 and T2 distance criteria result in very large neighbourhoods, especially on scale-free and NLPA graphs due to the presence of *hub* agents with a large number of neighbours. We also observe that K5 criterion generates a large number of neighbourhoods, specially in scale-free and NLPA networks due to the presence of hub nodes (e.g. the average number of neighbourhoods per agent in NLPA graphs is 11366).

From this analysis we can conclude that k-size generates a potentially huge number of neighborhoods of limited size and t-distance generates a limited number of potentially huge neighborhoods. To overcome this, we introduce a new type of regions, namely size-bounded distance regions, which include a limited number of bounded size neighborhoods.

6.3.2 Size-bounded distance optimality

Our aim at formulating the size-bounded distance criterion is to provide an alternative trade-off to size and distance, being more aware of the complexity of the regions they generate.

Let $T(x_i, x_j)$ be the distance between two variables in the constraint graph. Let $\Omega_t(x_i) = \{x_i | T(x_i, x_j) \leq t\}$ be the t-distance neighbourhood centered on variable x_i. Then, the s-size-bounded-distance neighbourhood is the largest t-distance region whose number of variables does not exceed the limit s. Formally, let $t(x_i) = \max\{t \text{ s.t. } |\Omega_t(x_i)| \leq s\}$ be the largest value for t such that $|\Omega_t(x_i)| \leq s$.

[2]As in (Kiekintveld et al., 2010; Pearce and Tambe, 2007) neighbourhoods of 5 variables that are not connected in the graph are discarded.

The s-size-bounded-distance neighbourhood centered on variable x_i is defined as $\Phi_s(x_i) = \Omega_{t(x_i)}(x_i)$.

For instance figure 6.4 (b)-(g) depicts 5-size-bounded distance neighbourhoods for agents x_0 to x_5 for the DCOP in figure 6.4 (a). Observe that agents can end exploring different distance levels in their neighbourhoods as a result of bounding their size to s. In our example, agents x_0, x_2, x_3 and x_5 explore their 2-distance neighbourhood with size 5 (figures 6.4 (b)(d)(e)(g)), whereas agents x_1 and x_4 are restricted to 1-distance neighbourhood with size 4 (figures 6.4 (c)(f)).

Now, the $s-$size-bounded distance region includes the s-size-bounded-distance neighbourhood of each agent $x_i \in \mathcal{X}$. Moreover, in order to ensure that all relations are covered, the s-size-bounded-distance region also includes a neighbourhood for every edge in the graph.

Note that in size-bounded distance optimality both the number of neighbourhoods and their size are limited. Now we can go back to table 6.5, to compare the number of regions and its size with the state-of-the-art criteria. In the last row we show the averages over 50 constraint graphs of the maximum neighbourhood size in a region (MaxS) and the number of neighbourhoods per agent (#) for 5-size-bounded-distance optimality (S5) for each type of graph. We observe that S5 is the only criterion that manages to keep the size of the region limited (to 5 agents) together with a reasonable number of neighbourhoods per agent (between 3 and 10 depending on the graph structure).

6.3.3 DALO for region optimality

The DALO algorithm is an asynchronous algorithm that starts with a random initial assignment and monotonically increases the solution quality by independently optimizing in each of the neighbourhoods that are created. As described in (Kiekintveld et al., 2010), DALO has three phases: initialization, optimization, and implementation.

During the initialization phase, agents distributedly create a set of neighbourhoods and assign each neighbourhood to a leader agent (the central node to minimize communication), which will be in charge of its optimization. After initialization, agents run in parallel the optimization and implementation phases for each assigned neighbourhood until stabilization.[3] During the optimization phase, each leader agent optimises by searching for a joint assignment of the variables in its neighborhoods that improves their reward. After optimizing, the leader agent runs the implementation phase trying to implement the new joint assignment found. Because neighbourhoods are optimised in parallel and a variable can appear in multiple neighbourhoods, the DALO implementation phase uses an asynchronous protocol based on a standard lock/commit pattern to ensure stability during implementation.

To use DALO with an arbitrary region $\mathcal{C}$, we focused on the initialization phase to modify how agents create the groups over which they optimise. Concretely, to allow DALO to search for a $\mathcal{C}$-optimal, during the initialization phase agents will distributedly generate the neighbourhoods in region $\mathcal{C}$. For example, to use DALO in the s-size-bounded distance region, each agent will iterate through various t-distance neighbourhoods by broadcasting at distance t, to determine the largest t-distance neighbourhood

[3] Stabilization is detected in DALO when no change applies after some number of iterations.

whose size does not exceed the threshold s.

After initialization, for the specific $\mathcal{C}$ region, the optimization and implementation phases are ran as specified in (Kiekintveld et al., 2010), independently of the region they use.

6.3.4 Empirical results

In this section we compare the results obtained by DALO using four different criteria: 5-size ($K5$), 1-distance ($T1$), 2-distance ($T2$), and 5-size bounded distance ($S5$) criteria.

We ran similar experimental settings to Kiekintveld et al. (Kiekintveld et al., 2010). We measured the performance of the extension of the DALO algorithm[4] described in section 6.3.3 when running over each one of the regions generated by the four criteria described above. Thus, we tested DALO for the four criteria over the different types of graphs described in section 6.3.1. All the graphs have 100 nodes, each one with density 4, meaning that on average each node has 4 neighbours. Moreover, variables' domain size is 10, and rewards are integers sampled from a distribution $U[0, 10000]$.

Besides graph types, we also considered different *Computation/Communication Ratios* (CCR) (Kiekintveld et al., 2010). The CCR setting defines the number of constraint assignments that may be evaluated at each communication step. For example, $CCR = 0.01$ allows each node to process up to 100 checks during a time step. We vary the setting of CCR in our experiments to test DALO in two settings with different relative cost for sending messages and computation, namely $CCR = 0.01$ and $CCR = 0.1$. In general, the larger the value of CCR, the higher the computation cost. Notice that, with respect to the experimental settings in (Kiekintveld et al., 2010), we discarded using $CCR = 0$. The rationale is that if $CCR = 0$, communication is infinitely more costly than computation, and hence the best strategy is computing the optimal by means of a fully centralized algorithm.

Figures 6.5 (a)-(f) plot the normalized solution quality of each algorithm along global time for each graph structure and CCR metric. The normalized solution quality is computed by: (1) subtracting the initial reward, as assessed by DALO for a given criterion, from the reward at a given global time; and (2) dividing the result by the best known reward obtained by DALO out of the four criteria. All results are averaged over 25 sample instances. In what follows, we compare the four criteria along two dimensions: (1) the final normalised solution quality; and (2) the convergence speed required to reach a good solution quality.

Regarding solution quality, the results vary depending on the value of CCR and the graph structure. On the one hand, in scenarios where computation is more costly ($CCR = 0.1$), overall $S5$ outperforms the rest of criteria. Although $T1$ is very competitive and its solution quality comes very close to that of $S5$ over random and scale free graphs, $S5$ significantly outperforms $T1$ on NLPA graphs. Moreover, both $S5$ and $T1$ largely outperform $K5$. The reason for the poor performance of $K5$ is that it generated neighbourhoods of fixed size. On the other hand, in scenarios where computation is cheaper ($CCR = 0.01$), the differences of final solution qualities between $S5$, $T1$, and $K5$ are not significant. There is an aspect though that deserves special attention. Notice

[4]We used the DALO code provided by the authors in `http://teamcore.usc.edu/dcop/`.

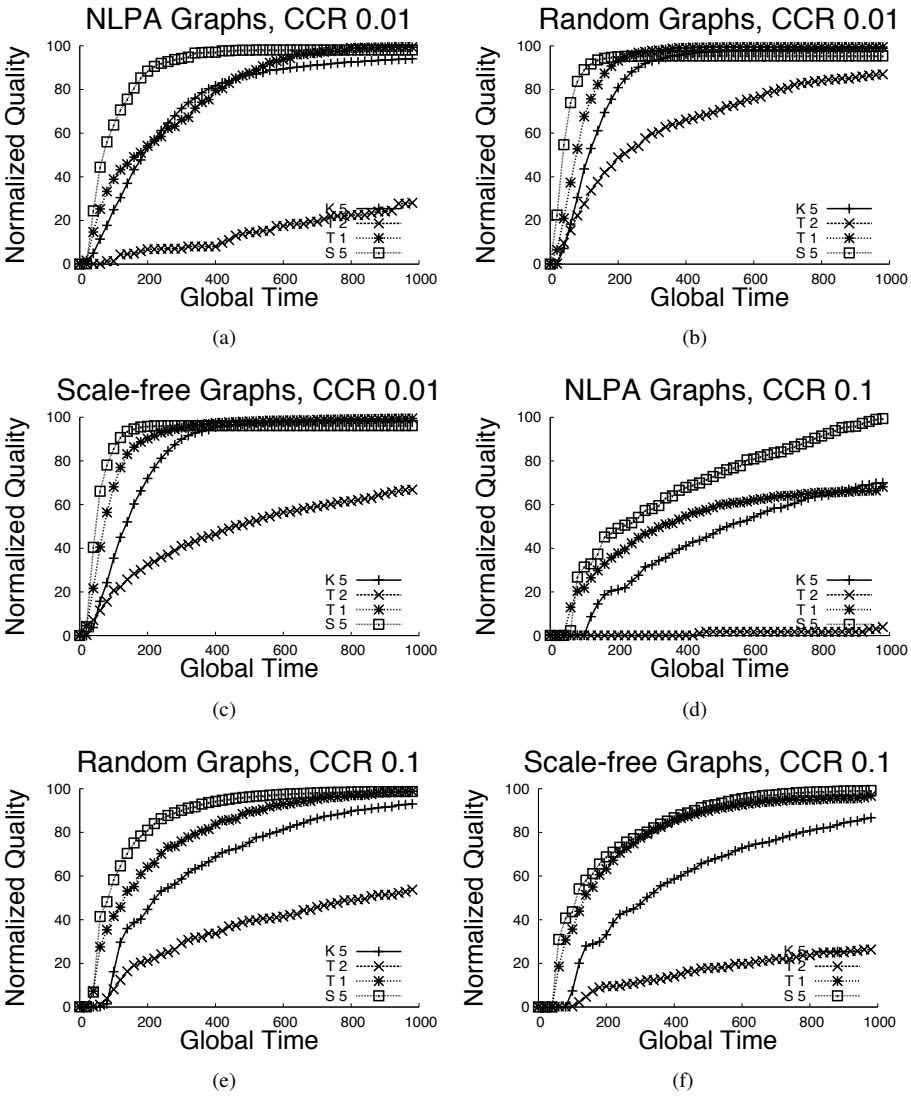


Figure 6.5: Experimental results comparing DALO for $K5$, $T1$, $T2$ and $S5$ regions.

that for all the test cases, the performance of DALO over $T2$ regions is much worse than the performance over the regions generated by the rest of criteria. We can explain this result by analysing the complexity of $T2$ regions as shown in table 6.1. Thus, we observe that $T2$ generates very large neighbourhoods that can not be optimised within the maximum global time (1000 global time steps). The solution quality degradation when handling $T2$ regions is particularly significant on scale-free and NLPA graphs because the criterion generates neighbourhoods whose size is close to the size of the

original problem (99 variables on average in NLPA graphs).

Regarding convergence speed, $S5$ regions help DALO converge to a high solution quality faster than the rest of regions. Likewise our analysis about solution quality above, $T1$ is again competitive with respect to $S5$, though $S5$ largely outperforms $T1$ on NLPA graphs. This is because, as observed in (Kiekintveld et al., 2010), NLPA graphs are characterized by large *hub* nodes with many connections that results in large neighbourhoods that take long for agents to optimise. Regarding $K5$, convergence speed is slower than that of $S5$ and $T1$ because each leader in DALO coordinates a neighbourhood of size 5, whereas the neighbourhoods for $S5$ and $T1$ may be smaller.

To summarise, our experimental results show that criteria that produce regions with large number of neighbourhoods or/and large neighbourhood sizes are not guaranteed to outperform criteria that produce less complex regions. In fact, overall the size-bounded distance criterion proposed in section 6.3.2 was able to outperform the rest of criteria by limiting the complexity of the regions that it generates.

6.4 Per-reward region optimal bounds

Sections 6.2.2 and 6.2.3 define two different mechanisms that assess bounds on any $\mathcal{C}$ optimum, independently of the DCOP rewards. As shown in (Bowring et al., 2008) for the particular case of k-size optimality, one can provide tighter bounds by assuming some reward structure. Along this line, this section shows how to extend region optimal bounds to exploit information about the reward structure of the problem, if available. Concretely, in the next sections we show how to tighten the region optimal bounds provided by the mechanisms in sections 6.2.2 and 6.2.3 by assuming:

- a ratio between the least minimum reward to the maximum reward among constraints, the so-called *minimum fraction reward* (section 6.4.1); and
- the knowledge of the minimum and maximum rewards per relation, the so-called *extreme relations rewards* (section 6.4.2).

Finally, section 6.4.3 characterises the gain on tightness obtained when exploiting the knowledge about these different reward structures.

6.4.1 Exploiting the minimum fraction reward

In this section we show how to tighten fine or coarse region optimal bounds defined in section 6.2 when we know that the minimum reward is a certain factor β ($0 < \beta \leq 1$) of the maximum reward of any relation. Thus, this refinement is a generalization of the improvements in tightness for k-size optimal bounds proposed in (Bowring et al., 2008) to apply them to any $\mathcal{C}$ optimum.

Extending fine region optimal guarantees to exploit knowledge about the minimum fraction reward

First, we show that assuming a minimum fraction reward β we can tighten the fine quality guarantees obtained by means of the mechanism described in section 6.2.2. In

order to obtain a tighter bound, we employ the set of partially covered relations.

Thus, instead of setting the values of all relations in the set of partially covered relations $P(A^k)$ to 0 as we did in equation 6.3, we can now exploit the knowledge that $r(x^{\alpha_k}) \geq \beta \cdot r(x^*)$. Then, for all $A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\}$, equation 6.2 can be rewritten as:

$$\sum_{r \in \mathcal{R}} r(x^{\mathcal{C}}) \geq \sum_{r \in T(C^\alpha)} r(x^*) + \sum_{r \in P(C^\alpha)} \beta \cdot r(x^*) + \sum_{r \in N(C^\alpha)} r(x^{\mathcal{C}}) \tag{6.9}$$

Notice that this is the only change we need incorporate the knowledge about the minimum fraction reward. Hence, from equation 6.9 we can also build an LP, following analogous operations as the ones detailed in section 6.2.2 and with the same number of variables. As an example, we turn back to figure 6.1 to assess the fine region optimal bound for the 2-size region depicted in figure 6.1(b) when assuming a minimum fraction reward β. With respect to the LP formulation in section 6.2.2 that is independent of the problem rewards, the right hand side of each constraint is modified to add the real variables related to the values of x^* for the partially covered relations multiplied by β. This results in the following LP formulation:

minimize $z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}}$
subject to
$y_{r_{01}} + y_{r_{13}} + y_{r_{23}} + y_{r_{02}} = 1$
and subject to:
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{01}} + \beta \cdot (y_{r_{13}} + y_{r_{02}}) + z_{r_{23}}$
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{02}} + \beta \cdot (y_{r_{01}} + y_{r_{23}}) + z_{r_{13}}$
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq \beta \cdot (y_{r_{01}} + y_{r_{13}} + y_{r_{23}} + y_{r_{02}})$
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq \beta \cdot (y_{r_{01}} + y_{r_{13}} + y_{r_{23}} + y_{r_{02}})$
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{13}} + \beta \cdot (y_{r_{01}} + y_{r_{23}}) + z_{r_{02}}$
$z_{r_{01}} + z_{r_{13}} + z_{r_{23}} + z_{r_{02}} \geq y_{r_{23}} + \beta \cdot (y_{r_{13}} + y_{r_{02}}) + z_{r_{01}}$

The solution of the LP is a tight bound on the quality of a region optimum for the graph structure represented by $\mathcal{R}$ and rewards with a minimum fraction reward of β. Thus, by solving the above LP with β set to $\frac{1}{2}$ we assess a region optimal bound $\delta = 2/3$. Notice that this per-reward bound is significantly higher than the bound assessed in section 6.2.2. Table 6.2 shows the fine per-minimum fraction reward for two reward structures with $\beta = \frac{1}{2}$. Observe that the fine per-minimum fraction reward bound for these reward structures, $\delta = 2/3$, doubles the fine reward-independent bound, $\delta = 1/3$.

Extending coarse region optimal guarantees to exploit knowledge of the minimum fraction reward

Now, we show that assuming a minimum fraction reward β we can also improve the coarse region optimal bounds introduced in section 6.2.3. Recall that each relation $r \in \mathcal{R}$, the number of neighbourhoods in region $\mathcal{C}$ that partially cover relation r are defined as $pc(r, \mathcal{C}) = |\mathcal{C}| - nc(r, \mathcal{C}) - cc(r, \mathcal{C})$. Then, the following proposition shows how to exploit the minimum fraction reward along with the partially covered relations to obtain a bound tighter than the one in equation 6.4.

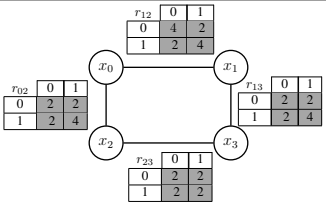
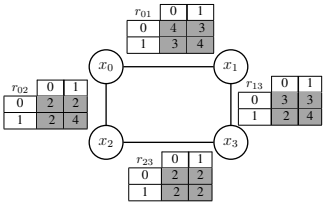
Reward Structure	Reward-independent		Minimum Fraction Reward			Per Extreme Rewards			
	Fine	Coarse	β	Fine	Coarse	L	U	Fine	Coarse
	$\frac{1}{3}$	$\frac{1}{5}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{5}$	8	16	$\frac{2}{3}$	$\frac{3}{5}$
	$\frac{1}{3}$	$\frac{1}{5}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{5}$	10	16	$\frac{3}{4}$	$\frac{7}{10}$

Table 6.2: Comparison of independent reward, per minimum fraction and per extreme reward bounds for the 2-size region of the DCOP constraint graph in figure 6.1(a) under different reward structure assumptions.

Proposition 9. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region and β the minimum fraction reward. If $x^{\mathcal{C}}$ is a $\mathcal{C}$ optimum, then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \left(\frac{cc_*}{|\mathcal{C}| - nc_*} + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \right) \mathcal{R}(x^*), \tag{6.10}$$

where $cc_* = \min_{r \in \mathcal{R}} cc(r, \mathcal{C})$, $nc_* = \min_{r \in \mathcal{R}} nc(r, \mathcal{C})$, $pc_* = \min_{r \in \mathcal{R}} pc(r, \mathcal{C})$, and x^* is the optimal assignment.

The proof for proposition 9 is analogous to the one for the general bound of equation 6.4 formulated in section 6.2.3, but without disregarding partially covered relations. For the sake of readability, the proof is provided in Appendix B.

Proposition 9 directly provides a simple algorithm to compute a bound. Given a region $\mathcal{C}$ and a graph structure, we can directly assess cc_*, pc_* and nc_* by computing $cc(r, \mathcal{C})$, $pc(r, \mathcal{C})$, and $nc(r, \mathcal{C})$ for each relation $r \in \mathcal{R}$ and taking the minimum. This will take time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$, which is linear in the number of relations of the DCOP and linear in the number of neighbourhoods in the region. As an example, now we turn back to figure 6.1 to assess the bound for the 2-size region $\mathcal{C}_2$ in figure 6.1(b) when assuming a minimum fraction reward $\beta = \frac{1}{2}$. Table 6.2 shows two reward structures for the DCOP constraint graph of figure 6.1(a) in which $\beta = \frac{1}{2}$. Given the relation r_{01} we assess the number of neighbourhoods that completely cover $\{x_0, x_1\}$ as

$cc(r_{01}, \mathcal{C}_2) = 1$ (the first neighbourhood), the number of neighbourhoods that partially cover $\{x_0, x_1\}$ as $pc(r_{01}, \mathcal{C}_2) = 4$ (from the second to the fifth neighborhoods) and the number of neighbourhoods that do not cover $\{x_0, x_1\}$ at all as $nc(r_{01}, \mathcal{C}_2) = 1$ (the sixth neighbourhood). After repeating the process for the rest of relations in the constraint graph, we obtain that $cc_* = 1$, $pc_* = 4$ and $nc_* = 1$, and hence $\frac{cc_*}{|\mathcal{C}|-nc_*} + \beta\frac{pc_*}{|\mathcal{C}|-nc_*} = \frac{1}{6-1} + \frac{1}{2} \cdot \frac{4}{6-1} = \frac{3}{5}$. As summarised in table 6.2 this leads to a significant improvement with respect to $\delta = \frac{1}{3}$ and $\delta = \frac{1}{5}$, the fine and coarse bound assessed in section 6.2.3 independently of the rewards. However, this bound is not tight either because it is lower than $\delta = \frac{2}{3}$, the per-reward fine bound assessed by means of the LP.

6.4.2 Exploiting the extreme relation rewards

In this section we show how to tighten any region optimal bound computed independently of the rewards by assuming a minimum and maximum rewards for each relation $r_V \in \mathcal{R}$, namely $l_{r_V} = \min_{d_V \in \mathcal{D}_V} r_V(d_V)$ and $u_{r_V} = \max_{d_V \in \mathcal{D}_V} r_V(d_V)$ respectively.

Proposition 10. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region and δ a region optimal bound independent of the rewards. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \frac{1}{U}\left((U - L) \cdot \delta + L\right) \cdot \mathcal{R}(x^*) \tag{6.11}$$

where $U = \sum_{r \in \mathcal{R}} u_r$, $L = \sum_{r \in \mathcal{R}} l_r$.

Proof of proposition 10 is provided in Appendix B.

Proposition 10 directly provides a constant-time method to tighten any region optimal bound δ that has been computed independently of the rewards by assuming that the extreme values of relations are known. Because proposition 10 does not make any assumption about how bound δ is calculated other than it is independent of the DCOP rewards, this improvement applies to both: (i) the fine region optimal bound introduced in section 6.2.2; and (ii) the coarse region optimal bounds introduced in section 6.2.3.

As an example, we turn back to table 6.2 to assess the per extreme reward bounds for the two reward structures of the DCOP in figure 6.1(a). As summarised in this table, the fine and coarse independent reward bounds assessed in section are $\delta = \frac{1}{3}$ and $\delta = 1/5$ respectively.

Next, using equation 6.11 we compute the per-extreme rewards bounds for these two reward structures.

First, consider the reward structure in the first row of table 6.2 in which each relation of the DCOP in figure 6.1(a) has a minimum reward of 2 and a maximum reward of 4. Thus, $U = 16$ and $L = 8$. Using now equation 6.11 with the fine bound $\delta = \frac{1}{3}$ we obtain $\delta = 1/16 \cdot ((16 - 8) \cdot 1/3 + 8) = 2/3$. In a similar way, using equation 6.11 with the coarse bound $\delta = 1/5$ we obtain $\delta = 1/16 \cdot ((16 - 8) \cdot \frac{1}{5} + 8) = 3/5$. Observe that the bounds we obtain by using knowledge on extreme relations rewards are the same as the bounds obtained using the minimum fraction reward.

Now, consider the reward structure in the second row of table 6.2 in which the DCOP in figure 6.1(a) has two relations with a minimum reward of 2 and a maximum

of 4, and two relations with a minimum reward of 3 and a maximum of 4. Thus, $U = 16$ and $L = 10$. Using now equation 6.11 with the fine bound $\delta = \frac{1}{3}$ we obtain $1/16 \cdot ((16 - 10) \cdot 1/3 + 10) = 3/4$. Similarly, using equation 6.11 with the coarse bound $\delta = 1/5$ we obtain $1/16 \cdot ((16 - 10) \cdot \frac{1}{5} + 10) = 7/10$. Observe that in this second scenario, exploiting the knowledge about the extreme rewards of relations leads to higher bounds than exploiting the knowledge about the minimum fraction reward.

6.4.3 Comparing per-reward region optimal bounds

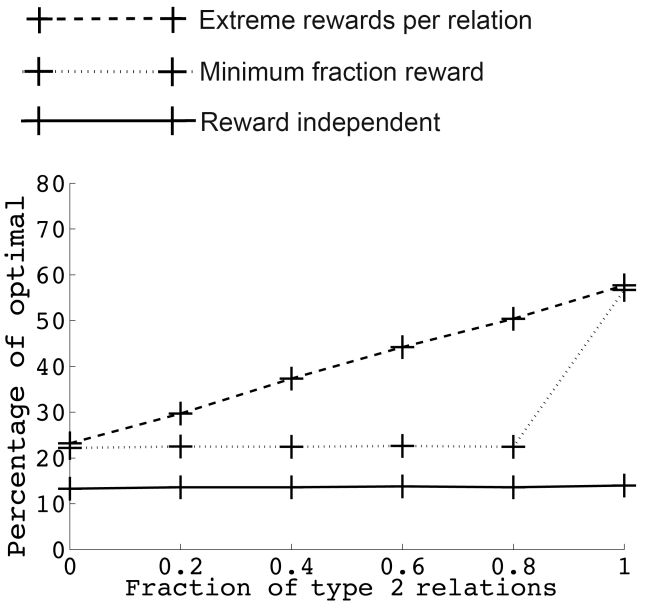
Since the more knowledge we exploit from a problem the tighter the quality guarantees, the per-reward region optimal guarantees proposed in the sections above are expected to be tighter than bounds defined in section 6.2. Furthermore, because not all the assumptions over the reward structure have the same level of specificity, exploiting the knowledge about the extreme rewards per relation is also expected to lead to tighter quality guarantees than only assuming a ratio between them.

Hence, with the aim of illustrating the tightness of region optimal quality guarantees depending on the knowledge degree about the reward structure, we produce: (i) empirical results that show the average-case improvement of fine region optimal bounds; and (ii) theoretical results that characterise the relations between faster region optimal bounds.

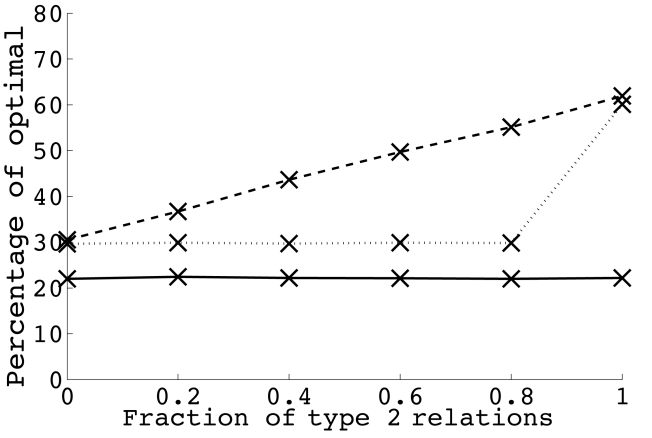
Comparing fine region optimal bounds

Figures 6.6(a)(b) show the values of fine region optimal bounds, calculated by means of the LP, defined as a percentage of the optimal, for random DCOPs with 100 agents and density 4 using as a criterion neighborhoods of size 3 and of distance 1 respectively. All results are averaged over 50 sample instances. Because, intuitively, the gain obtained by exploiting the knowledge about the extreme rewards per relation with respect to the minimum fraction reward varies with the heterogeneity of the reward structure, we generate DCOPs with two types of relations: (1) type 1, relations whose rewards are integers drawn from a uniform distribution $U[2500, 10000]$; and (2) type 2, relations whose rewards are integers drawn from a uniform distribution $U[5000, 10000]$.

The dotted lines show the per-reward bounds when exploiting the the minimum fraction reward. The dashed lines show the per-reward bounds when exploiting the knowledge of the extreme relation rewards. The solid lines show the region optimal bounds as presented in section 6.2.2, that apply to any reward structure. The x-axis represents the fraction of type 2 relations with respect to type 1 relations. First of all, observe that, independently of the particular knowledge exploited, per-reward bounds are significantly higher than the reward-independent bound. For example, in figure 6.6 (a) the reward independent bound is around 12% whereas per-reward bounds are around 22% when using only type 1 relations (x-axis $= 0$) and around 60% when using only type 2 relations (x-axis $= 1$). Moreover, it is worth noting that when all relations are of the same type, independently of the knowledge exploited about the reward, both per-reward bounds are very close. In contrast, in mixed instances, when both types of relations are present, the graphs show that minimum fraction reward bounds can not



(b) 3-size bounds



(c) 1-distance bounds

Figure 6.6: Per-reward bounds on 100 agent random DCOPs with density 4 using as a criterion: (a) size 3 and (b) distance 1.

improve the bound further by taking advantage of type 2 relations. Indeed, when exploiting the minimum fraction reward, we do not obtain a significant improvement with the introduction of relations of type 2 until relations of type 1 disappear. In contrast, when exploiting knowledge about extreme relation rewards, the bound progressively gets higher as the fraction of relations of type 2 increases.

In summary, these results show that exploiting more knowledge about the reward structure helps obtain tighter bounds for a wide range of reward distributions, particularly for heterogeneous distributions composed of rewards of different kinds.

Comparing coarse region optimal quality guarantees

On the one hand, notice that the coarse per-reward guarantees assessed by equations 6.10 and 6.11 are guaranteed to be greater than or, in the worst-case, equal to the fine reward-independent guarantees assessed by means of equation 6.4.

On the other hand, we are interested in comparing per-reward coarse quality guarantees when exploiting different assumptions over the reward structure. With this aim, next we prove that coarse quality guarantees that exploit knowledge about extreme rewards per relation are tighter than those exploiting the minimum fraction bound.

Proposition 11. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region, β the minimum fraction reward and $\delta = \frac{cc_*}{|\mathcal{C}|-nc_*}$, then*

$$\delta + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \leq \frac{1}{U}\left((U - L) \cdot \delta + L\right) \tag{6.12}$$

where $U = \sum_{r \in \mathcal{R}} u_r$, $L = \sum_{r \in \mathcal{R}} l_r$.

For the sake of readability, proof of proposition 11 is included in Appendix B.

Notice that the left handside of equation 6.12 corresponds to the coarse guarantees in equation 6.10 that exploit the minimum fraction reward, whereas the right handside corresponds to those in equation 6.11 that exploit extreme rewards per relation. Therefore, the purpose of proposition 11 is to formally prove that exploiting more knowledge about the reward structure, such as the extreme rewards per relation, helps obtain tighter coarse bounds.

6.5 Conclusions

As discussed above, k-size and t-distance optimality allow to compute guarantees of local optima in regions defined by size and distance respectively. This chapter overcomes this limitation by defining region optimality, a flexible framework that provides quality guarantees for optima in regions characterised by any arbitrary criterion.

With this aim, we contributed with: (i) a formal definition of region optimality, namely of local optimality in some arbitrary region; and (ii) quality guarantees for region optimal solutions that exploit the knowledge about the graph structure, if available. Regarding quality guarantees, we defined two methods with different computational costs: (i) a first one, based on solving an LP, that guarantees tightness; and (ii) a second one that requires linear time but does not ensure tightness. Moreover, we proved that k-size and t-distance optimality bounds (Pearce and Tambe, 2007; Kiekintveld et al., 2010) are particular instances of region optimal bounds.

Our empirical results show that approximate DCOP solving can benefit from exploring the larger space of criteria provided by region optimality. With that purpose, we

formulated $\mathcal{C}$-DALO algorithm, which enables an empirical evaluation of the average-case performance of different criteria.

Finally, in the last part of this chapter we show how to extend region optimal bounds to exploit a-priori knowledge of the reward structure of the problem, if available. To that end, we define reward-dependent bounds that can exploit as prior knowledge: (i) the ratio between the least minimum reward to the maximum reward among relations (minimum fraction reward); (ii) the extreme (minimum and maximum) rewards per relation. Empirical results show that exploiting knowledge about the reward structure leads to significantly tighter region optimal quality guarantees. Moreover, we also show that exploiting more detailed knowledge by assuming that extreme rewards per relation are known, we obtain tighter guarantees than only assuming knowledge about a minimum fraction reward.

Empirical results the average case performance of $\mathcal{C}$-DALO when employing size-bounded-distance criterion, leads to better solution qualities, outperforming k-size and t-distance criteria.

Figure 6.3 shows the resulting DCOP landscape after incorporating the aforementioned contributions of this chapter. Notice that the k-size optimal and t-distance optimal approaches are now unified under the region optimality framework. Moreover, now the landscape includes $\mathcal{C}$-DALO as a generic region optimal algorithm that returns local optima based on arbitrary criteria.

This chapter, with the extension of DALO algorithm to handle arbitrary criteria, proved the existence of an efficient region optimal DCOP algorithm for the whole space of criteria. However, given the algorithmic-independence of the region optimality framework, it does not imply that there cannot exist other efficient region optimal algorithms. Along this line, in the next chapter we employ the region optimal framework to provide quality guarantees on the solutions returned by the Max-Sum algorithm.

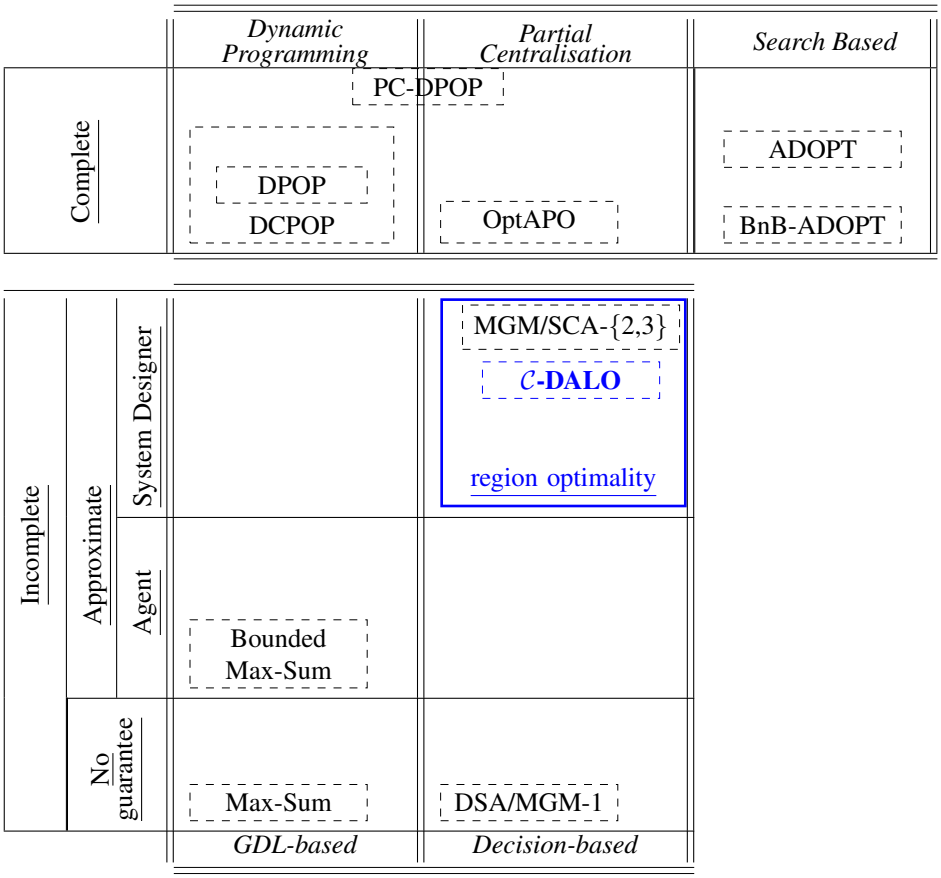


Table 6.3: DCOP algorithms landscape after region optimality. Contributions of this chapter are highlighted in blue/bold. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

Chapter 7

Max-sum as a region optimal algorithm

The region optimal framework presented in chapter 6 defines quality guarantees over region optimal solutions, independently of the particular algorithm employed to find them. Thus, region optimality is open to the usage of further algorithms besides $\mathcal{C}$-DALO. In this chapter we deal with this issue by proving region optimality as a valuable tool to bound at design time, the solutions of one of the leading incomplete DCOP algorithms, the Max-Sum algorithm.

As reviewed in chapter 3, Max-Sum corresponds to the iterative, approximate version of GDL and hence, it is equivalent to the well-known Loopy Belief Propagation (Pearl, 1988) or Max-Product (Aji and McEliece, 2000) algorithms. Therefore, unlike $\mathcal{C}$-DALO where agents optimize their decisions in groups, Max-Sum follows a GDL approach in which individual agents exchange messages about the particular utility to set their decisions variables to a particular state. Max-Sum algorithm is not restricted to DCOP solving, and indeed, is one of the most popular techniques to find the most likely joint variable assignment in graphical models, such as Markov Random Fields (MRFs). Thus, in addition to Multi-Agent coordination, Max-Sum has been successfully applied to a wide variety of applications such as image understanding (Tappen and Freeman, 2003), error correcting codes (Feldman et al., 2005) and protein folding (Yanover and Weiss, 2002), to name a few. In all these domains, the popularity of Max-sum stems for its good empirical performance on general MRFs (Aji et al., 1998; Frey et al., 2001a,b; Weiss, 2000; Farinelli et al., 2008) although this behaviour is not well understood because it comes with few theoretical guarantees.

Concretely, Max-Sum is known to be correct in acyclic and single-cycle graph structures (Weiss, 2000), although convergence is only guaranteed in the acyclic case. Recently, some works have established that Max-Sum is guarantee to return the optimal solution, if it converges, on graphical models corresponding to some specific problems, namely: (i) weighted b-matching problems (Bayati et al., 2007; Sanghavi et al., 2007); (ii) maximum weight independent set problems (Sanghavi et al., 2008); or (iii) problems whose equivalent NAND Markov random field (NMRF) is a perfect graph (Jebara,

2009). For weighted b-matching problems with a bipartite structure, Huang and Jebara (Huang and Jebara, 2007) establish that Max-Sum always converges to the optimum.

Despite the guarantees provided in these particular cases, for general problems little is known on the quality of the solutions that Max-Sum achieves on convergence. To the best of our knowledge, the only result in this line is the work of Wainwright et al. (Wainwright et al., 2004) where, given any arbitrary MRF, authors derive an upper bound on the absolute error of the Max-Sum solution. However, this bound is calculated on Max-Sum convergence and depends on the particular problem instance. Hence, this bound can not be used at design time. Moreover, it can neither be used by agents at runtime. This is because Max-Sum is not guaranteed to converge in a linear number of iterations (not even guaranteed to converge) and hence, this bound is not available anytime during the Max-Sum execution.

Against this background, in this chapter we are the first to provide quality guarantees for Max-Sum solutions on convergence in general settings at design time. To this end, we define worst-case bounds on the quality of any Max-Sum solution, independently of the problem. In addition to these problem-independent bounds, we also assess guarantees for a collection of specific graph structures whereas illustrating how to compute guarantees for other graph structures.

Our results build upon two main components: (i) the characterization of any Max-Sum solution as neighbourhood maximum in a specific region of the MRF, the so-called *Single Loops and Trees* (SLT) region (Weiss and Freeman, 2001); and (ii) the worst-case bounds on the quality of any region optimum provided by the region optimal framework introduced in chapter 6. Therefore, the main contribution of this chapter is to combine these two results to bound the quality of Max-Sum solutions applying the region optimal framework to the region characterised in (Weiss and Freeman, 2001).

This chapter is organised as follows. Section 7.1 provides some background on the Max-Sum algorithm and on the characterisation of any Max-Sum solution as a region optimum. Next, sections 7.2 and 7.3 show how to define: (i) tight guarantees for Max-Sum assuming any arbitrary graph structure (section 7.3); and (ii) coarse guarantees that can be assessed on constant time for specific graph structures (section 7.2). Finally, section 7.4 summarises the contributions of this chapter and draws some final remarks.

7.1 Background: the Max-Sum algorithm

This section offers an overview of the Max-Sum algorithm, complementary to the review in chapter 3.

We start by describing the Max-Sum operation when searching for the MAP assignment in graphical models, such as Markov Random Fields (MRFs). Afterwards, we review the characterisation of a Max-Sum solution as a region optimum as detailed in (Weiss and Freeman, 2001).

7.1.1 Max-Sum in Pairwise Markov Random Fields

Max-Sum is one of the most popular techniques to find the most likely joint variable assignment in graphical models, namely the maximum a posteriori (MAP) assignment.

In this chapter we formulate our contribution in terms of the MAP because it subsumes DCOP. Hence, next we formalise the MAP over an MRF.

A discrete pairwise Markov Random Field (MRF) is an undirected graphical model where each interaction is specified by a discrete potential function, defined on a single variable or a pair of variables. The structure of an MRF defines a graph $\mathcal{G} = \langle \mathcal{X}, E \rangle$, whose nodes $\mathcal{X}$ stand for discrete variables, and whose edges E represent interactions between nodes. Then, a pairwise MRF contains a unary potential function Ψ_i for each variable node $x_i \in \mathcal{X}$, and a pairwise potential function Ψ_{ij} for each edge $(i,j) \in E$; the joint probability distribution of the MRF assumes the following form:

$$\begin{aligned} P(d) &= \frac{1}{Z} \prod_{x_i \in \mathcal{X}} \Psi_i(d_i) \prod_{(i,j) \in E} \Psi_{ij}(d_i, d_j) \\ &= \frac{1}{Z} \exp\left(\sum_{x_i \in \mathcal{X}} \theta_i(d_i) + \sum_{(i,j) \in E} \theta_{ij}(d_i, d_j) \right) \\ &= \frac{1}{Z} \exp\left(\theta(d)\right), \end{aligned} \tag{7.1}$$

where d is an element of the joint domain space $\mathcal{D}$, d_i is the projection of d over x_i, Z is a normalization constant, and θ_i, θ_{ij} stand for the logarithms of the strictly positive potentials Ψ_i, Ψ_{ij}.

Within this setting, the classical *maximum a posteriori* (MAP) corresponds to finding the most likely configuration for the joint probability distribution P in equation 7.1. In more formal terms, the most likely (MAP) configuration x^* is given by:

$$\begin{aligned} x^{MAP} &\triangleq arg \max_{d \in \mathcal{D}} \left[\prod_{x_i \in \mathcal{X}} \Psi_i(d_i) \prod_{(i,j) \in E} \Psi_{ij}(d_i, d_j) \right] \\ &\triangleq arg \max_{d \in \mathcal{D}} \left[\sum_{x_i \in \mathcal{X}} \theta_i(d_i) + \sum_{(i,j) \in E} \theta_{ij}(d_i, d_j) \right], \end{aligned} \tag{7.2}$$

where d is an element of the joint domain space $\mathcal{D}$.

Note that the MAP configuration may not be unique, that is, there may be multiple configurations that attain the maximum in equation 7.1. In this chapter we assume that: (i) there is a unique MAP assignment (as assumed in (Weiss and Freeman, 2001)); and (ii) all potentials θ_i and θ_{ij} are non-negative.

Recall that, as described in chapter 2, solving a DCOP aims to assess a configuration x^* that maximises the DCOP utility function R. Therefore, observe that there is a the straightforward mapping between finding a MAP in a probabilistic distribution P and finding the optimal configuration x^* in a DCOP. Concretely, when setting each unary and binary potential θ_i and θ_{ij} in distribution P as the DCOP relations r_i and r_{ij} respectively, both problems are equivalent, with exception of the distribution requirement of DCOPs. Thus, throughout the rest of this chapter we will denote x^{MAP} as x^*. As explained in chapter 2, DCOP variables are assigned to agents, which are required to assess their optimal values in a distributed way. However, as discussed in (Farinelli

et al., 2008), Max-Sum is an iterative, local message passing algorithm, whose update rules can be directly implemented in a distributed way.

Concretely, the standard update rules for Max-Sum to find the MAP assignment in a discrete MRF as specified by equation 7.2 are the following:

$$\mu_{ij}(d_j) = \alpha_{ij} + \max_{d_i \in \mathcal{D}_i} \left[\theta_i(d_i) + \theta_{ij}(d_i, d_j) + \sum_{x_k \in N(x_i) \setminus x_j} \mu_{ki}(d_i) \right], \tag{7.3}$$

$$b_i(d_i) = \theta_i(d_i) + \sum_{x_k \in N(x_i)} \mu_{ki}(d_i), \tag{7.4}$$

where α_{ij} is a normalization constant, and $N(x_i)$ returns the set of variables that are connected to x_i, μ_{ij} is a message from x_i to x_j and b_i is the approximation of max-marginal over x_i. At the first iteration all messages are initialised to constant functions[1]. At each following iteration, each variable x_i aggregates all incoming messages and computes the belief b_i, which is then used to obtain the Max-Sum assignment $x^{MS} = \{x_i^{MS} | x_i \in \mathcal{X}\}$ where $x_i^{MS} = arg\max_{d_i \in \mathcal{D}_i} b_i(d_i)$.

The convergence of Max-Sum is usually characterized considering fixed points for the message update rules, i.e. when all the messages exchanged are equal to the last iteration. With a slight abuse of notation, through the rest of this chapter we use x^{MS} to denote the Max-Sum assignment obtained on convergence and we will refer to it simply as Max-Sum solution.

Now, the Max-Sum algorithm is known to be optimal over acyclic and single-cycle graphs. Unfortunately, on general graphs the aggregation of messages flowing into each variable only represents an approximate solution to the maximization problem. Nonetheless, it is possible to characterise the solution obtained by Max-Sum as we discuss below.

7.1.2 Region optimal characterisation of Max-Sum solutions

In (Weiss and Freeman, 2001), Weiss et al. characterize how well Max-Sum approximates the MAP assignment. In particular, they find the conditions for a Max-Sum solution x^{MS} to be a region optimum, namely greater than all other assignments in a specific large region around x^{MS}. Weiss et al. introduce the notion of *Single Loops and Trees* (SLT) region to characterise the assignments in such region.

Definition 21 (SLT region)**.** *An SLT-region of x in $\mathcal{G}$ includes all assignments x' that can be obtained from x by: (i) choosing an arbitrary subset $V \subseteq \mathcal{X}$ such that its vertex-induced subgraph contains at most one cycle per connected component; (ii) assigning arbitrary values to the variables in V while keeping the assignment to the other variables as in x.*

[1]The constant used is the neutral value that depends on the nature of the problem. For example, for utility maximisation the constant is 0 whereas for probabilistic settings is 1.

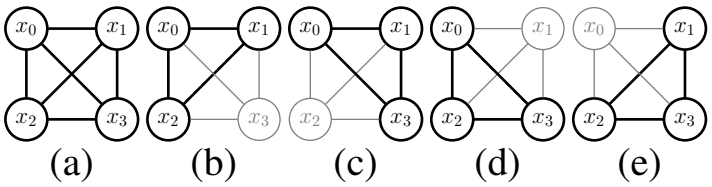


Figure 7.1: (a) 4-complete graph and (b)-(e) sets of variables covered by the SLT-region.

Hence, we say that an assignment x^{SLT} is SLT optimal if it is greater than any other assignment in its SLT region. Finally, the main result in (Weiss and Freeman, 2001) is the characterisation of any Max-Sum solution as an SLT optimum. Figures 7.1(b)-(e) illustrate examples of assignments in the SLT region in the complete graph of figure 7.1(a), here boldfaced nodes stand for variables that vary the assignment with respect to x^{SLT}.

In the next sections we use region optimality to bound Max-Sum solutions in the SLT regions.

7.2 Fine Single Loops and Trees region optimal bounds

Since SLT is a region we can directly directly compute a fine region optimal bound on any SLT optimum by means of the mechanism introduced in chapter 6 (section 6.2.2).

As explained in chapter 6, this problem can be transformed into an LP whose solution is a tight bound for any MRF with the specific graph structure $\mathcal{G}$. However, as argued in (Weiss and Freeman, 2001), in many graphs the SLT region is exponentially large. Hence, the complexity of generating the LP is, in general, exponential to the number of variables of the MRF. This complexity is not surprising since assessing all the neighbourhoods in the SLT region requires generating all the vertex-induced trees and vertex-induced single cycle graphs of $\mathcal{G}$.

Although this method has the advantage of providing a tight bound for MRFs with any arbitrary structure, it does not scale well with the size of MRFs. Nonetheless, in the next section we show how to use the coarse method introduced in chapter 6 to bound Max-Sum solutions for any MRF or for MRFs with particular structures. For example, we can compute in constant time a bound for MRFs with a 2-D grid structure, even when, as shown in (Weiss and Freeman, 2001), the SLT region in this case is exponential to the number of variables in the MRF.

7.3 Coarse Single Loops and Trees region optimal bounds

In this section we show that by means of the coarse region optimal qualities introduced in chapter 6 (section 6.2.3) we can assess SLT-bounds for MRFs with arbitrary and specific structures in linear time.

The main idea is that by virtue of the characterization of any Max-Sum fixed point assignment as SLT-optimal, we can select any region $\mathcal{C}$ composed of a combination of single cycles and trees of our graph. Such region can be used to compute the corresponding $\mathcal{C}$-optimal bound by means of proposition 6 formulated in chapter 6.

We start by proving that coarse SLT region optimal bounds for a given graph apply to its subgraphs. Then, we find that the bound for the complete graph is problem-independent, and hence, it applies to any MRF independently of its graph structure and parameters. Afterwards, we define bounds tighter than the problem-independent bound for MRFs with specific structures.

7.3.1 Problem-independent Single Loops and Trees region optimal bounds

Next we show that the coarse SLT region optimal bounds for a given graph can be applied to any of its subgraphs.

Proposition 12. *Let $\mathcal{G} = \langle \mathcal{X}, E \rangle$ be a graphical model and $\mathcal{C}$ the SLT region of $\mathcal{G}$. Let $\mathcal{G}' = \langle \mathcal{X}', E' \rangle$ be a subgraph of $\mathcal{G}$. Then the coarse region optimal bound defined in chapter 6 for $\mathcal{G}$, namely:*

$$\theta(x^{\mathcal{C}}) \geq \frac{cc_*}{|\mathcal{C}| - nc_*}\theta(x^*) \tag{7.5}$$

where $cc_* = min_{S \in E}\, cc(S, \mathcal{C})$, $nc_* = min_{S \in E}\, nc(S, \mathcal{C})$, and x^* is the MAP assignment holds for any SLT optimal assignment in $\mathcal{G}'$.

For the sake of readability, the proof for proposition 12 is omitted here, but can be consulted in Appendix C. Next, we outline a sketch of the proof.

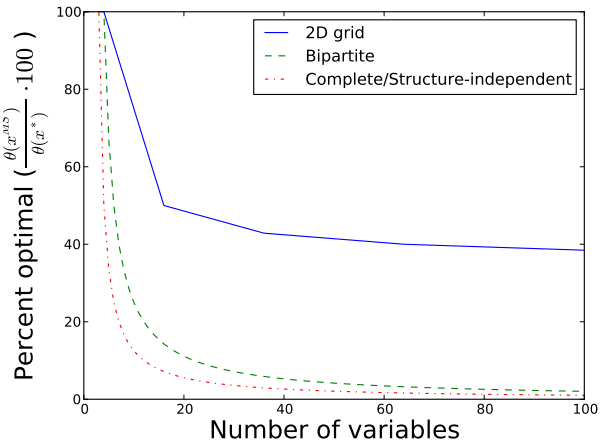
Sketch of the proof. We can compose a region $\mathcal{C}'$ containing the same elements as $\mathcal{C}$ but removing those variables which are not contained in $\mathcal{X}'$. Note that SLT-optimality on $\mathcal{G}'$ guarantees optimality in each element of $\mathcal{C}'$. Observe that the bound obtained by applying equation 6.4 to $\mathcal{C}'$ is greater or equal than the bound obtained for $\mathcal{C}$. Hence, the bound for $\mathcal{G}$ applies also to $\mathcal{G}'$.

A direct conclusion of proposition 12 is that any bound based on the SLT-region of a complete graph of n variables can be directly applied to any subgraph of n or fewer variables regardless of its structure. In what follows we assess the bound for a complete graph.

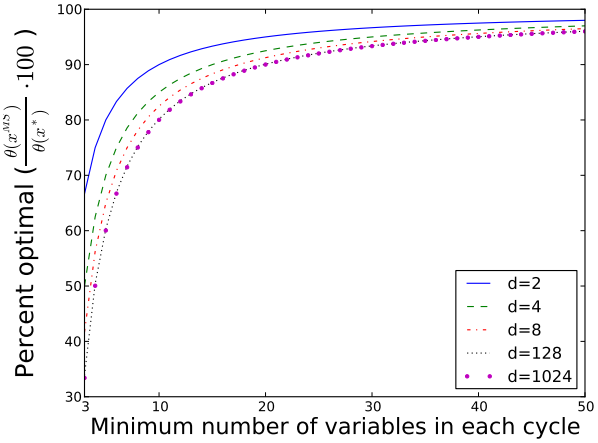
Proposition 13. *Let $\mathcal{G} = \langle \mathcal{X}, E \rangle$ be a complete MRF. For any Max-Sum solution x^{MS},*

$$\theta(x^{MS}) \geq \frac{1}{|\mathcal{X}| - 2} \cdot \theta(x^*). \tag{7.6}$$

Proof. Let $\mathcal{C}$ be a region containing every possible combination of three variables in $\mathcal{X}$. Every set of three variables is part of the SLT-region because it can contain at most one cycle. Then, the development in the proof of proposition 8 for k-size region optimality given in chapter 6 (section 6.2.4) can be applied here for $k = 3$ to obtain equation 7.6. □



(a) Bounds on complete, bipartite and 2-D grid structures when varying the number of variables.



(b) Bounds on MRFs with variable-disjoint cycles when varying the number of cycles (d) and their size (x-axis).

Figure 7.2: Percent SLT-region optimal bounds for Max-Sum solutions in MRF with specific graph structures.

Notice that any set of four variables in a complete graph has more than one cycle, and hence it is not contained in the SLT region. Thus, the SLT-region of a complete graph contains the same sets as the 3-size optimal region, and hence, any Max-Sum solution in a complete graph is 3-size region optimal.

Corollary 1. *For any MRF, any Max-Sum solution x^{MS} satisfies equation 7.6.*

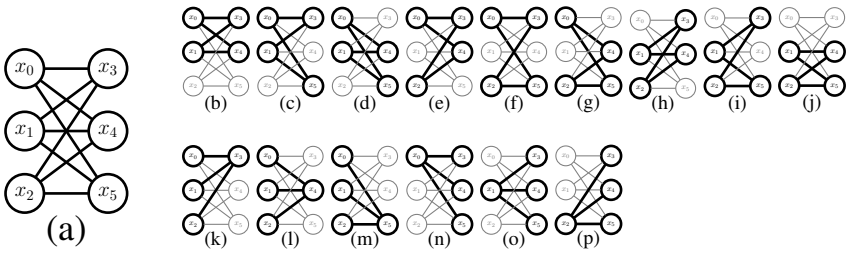


Figure 7.3: Example of (a) a 3-3 bipartite graph and (b)-(p) sets of variables covered by the SLT region.

Since any graph can be seen as a subgraph of the complete graph with the same number of variables, the corollary is straightforward given propositions 12 and 13.

Figure 7.2(a) plots this problem-independent bound when varying the number of variables. Observe that it rapidly decreases with the number of variables and it is only significant on very small MRFs. In the next section, we show how to exploit the knowledge of the structure of an MRF to improve the bound's significance.

7.3.2 Per-structure coarse SLT region optimal bounds

In this section we show that for MRFs with specific structures, it is possible to provide bounds much tighter than the problem-independent bound provided by corollary 1. These structures include, but are not limited to, bipartite graphs, 2-D grids, and variable-disjoint cycle graphs.

Bipartite graphs

We define the $\mathcal{C}$-optimal bound of equation 7.5 for any Max-Sum fixed point assignment in an n-m bipartite MRF. An n-m bipartite MRF is a graph whose vertexes can be divided into two disjoint sets, one with n variables and another one with m variables, such that the n variables in the first set are connected to the m variables in the second set. Figure 7.3(a) depicts a 3-3 bipartite MRF.

Proposition 14. *For any MRF with n-m bipartite structure where $m \geq n$, and for any Max-Sum solution x^{MS} we have that:*

$$\theta(x^{MS}) \geq b(n,m) \cdot \theta(x^*) \qquad b(n,m) = \begin{cases} \frac{1}{n} & m \geq n+3 \\ \frac{2}{n+m-2} & m < n+3 \end{cases} \tag{7.7}$$

For the sake of readability, the proof for proposition 14 is omitted here, but can be consulted in Appendix C. Next, we outline a sketch of the proof.

Proof. Let $\mathcal{C}^A$ be a region including one out of the n variables and all of the m variables (in figure 7.3, elements (n)-(p)). Since the elements of this region are trees, we can guarantee optimality on them. The number of elements of the region is $|\mathcal{C}^A| = n$. It is

clear that each edge in the graph is completely covered by one of the elements of $\mathcal{C}^A$, and hence $cc_* = 1$. Furthermore, every edge is partially covered, since all of the m variables are present in every element, and hence $nc_* = 0$. Applying equation 6.4 gives the bound $1/n$.

Alternatively, we can define a region $\mathcal{C}^B$ formed by taking sets of four variables, two from each set. Since the elements of $\mathcal{C}^B$ are single-cycle graphs (in figure 7.3, elements (b)-(j)), we can guarantee optimality on them. Applying proposition 6, we obtain the bound $\frac{2}{n+m-2}$. Observe that $\frac{2}{n+m-2} > \frac{1}{n}$ when $m < n+3$, and so equation 7.7 holds.

□

Example 7.3.2.1. *Consider the 3-3 bipartite MRF of figure 7.3(a). Figures 7.3(b)-(j) show the elements in the region $\mathcal{C}^B$ composed of sets of four variables, two from each side. Therefore $|\mathcal{C}^B|$ is 9. Then, for any edge $(i, j) \in E$ there are 4 sets in $\mathcal{C}^B$ that contain its two variables. For example, the edge that links the upper left variable (x_0) and the upper right variable (x_3) is included in the subgraphs of figures 7.3(b), (c), (e) and (f). Moreover, for any edge $(i, j) \in E$ there is a single element in $\mathcal{C}^B$ that does not cover it at all. For example, the only graph that does not include neither x_0 nor x_3 is the graph of figure 7.3(j). Thus, the bound is $4/(9-1) = 1/2$.*

Figure 7.2(a) plots the bound of equation 7.7 for bipartite graphs when varying the number of variables. Note that although, also in this case, the value of the bound rapidly decreases with the number of variables, it is twice the value of the problem-independent bound.

Two-dimensional (2-D) grids

Next, we define the $\mathcal{C}$-optimal bound of equation 7.5 for any Max-Sum solution in a 2-D grid MRF. An n-grid structure stands for a graph with n rows and n columns where each variable has 4 neighbours. Figure 7.4 (a) depicts a 4-grid MRF.

Proposition 15. *For any MRF with an n grid structure where n is an even number, for any Max-Sum solution x^{MS} we have that*

$$\theta(x^{MS}) \geq \frac{n}{3n-4} \cdot \theta(x^*) \tag{7.8}$$

Next, we outline a sketch of the proof for proposition 15. Details can be consulted in Appendix C.

Proof. We can partition columns in pairs joining column 1 with column $(n/2)+1$, column 2 with column $(n/2)+2$ and so on.

We can partition rows in the same way. Let $\mathcal{C}$ be a region where each element contains the vertexes in a pair of rows at distance $\frac{n}{2}$ together with those in a pair of columns at distance $\frac{n}{2}$. Note that optimality is guaranteed in each $C^\alpha \in \mathcal{C}$ because variables in two non-consecutive rows and two non-consecutive columns create a single-cycle graph. Since we take every possible combination, $|\mathcal{C}| = (\frac{n}{2})^2$. Each edge is completely covered by $\frac{n}{2}$ elements and hence $cc_* = \frac{n}{2}$. Finally, for each edge (i, j), there are

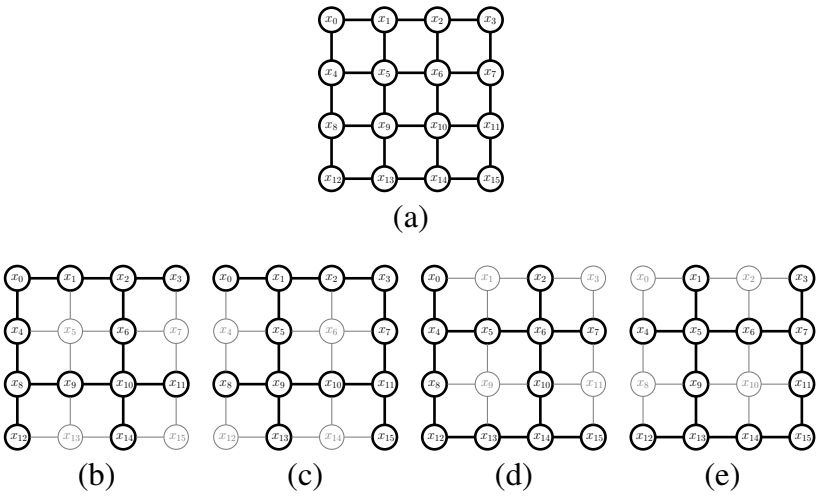


Figure 7.4: Example of (a) a 4-grid graph and (b)-(e) sets of variables covered by the SLT-region.

$nc_* = (\frac{n}{2} - 1)(\frac{n}{2} - 2)$ elements of $\mathcal{C}$ that do not cover $\{x_i, x_j\}$ at all. Substituting these values into equation 6.4 leads to equation 7.8. □

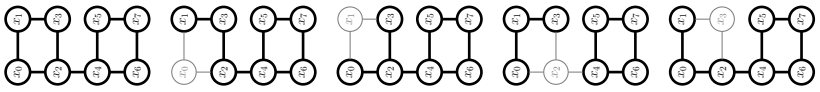
Example 7.3.2.2. *Consider the 4-grid MRF of figure 7.4 (a). Figures 7.4 (b)-(e) show the vertex-induced subgraphs for each set of vertexes in the region $\mathcal{C}$ formed by the combination of any pairs of rows in $\{(1,3),(2,4)\}$ and pair of columns in $\{(1,3),(2,4)\}$. Therefore $|\mathcal{C}| = 4$. Then, for any edge $(i,j) \in E$ there are 2 sets that contain its two variables. For example, the edge that links the two first variables in the first row, namely x_0 and x_1, is included in the subgraphs of figures (a) and (b). Moreover, for any edge $(i,j) \in E$ there is no set that contains no variable from $\{x_i, x_j\}$. Thus, the bound is* $1/2$.

Figure 7.2(a) plots the bound for 2-D grids when varying the number of variables. Note that when compared with the bound for complete and bipartite structures, the bound for 2-D grids decreases smoothly and tends to stabilize as the number of variables increases. In fact, observe that by equation 7.8, the bound for 2-D grids is never less that 1/3 independently of the grid size.

Variable-disjoint cycle graphs

In this section we assess a bound for MRFs composed of a set of variable-disjoint cycles, namely of cycles that do not share any variable.

A common pattern shared by the bounds assessed so far is that they decrease as the number of variables of an MRF grows. This section introduces an example showing that there are specific structures for which we obtain high quality guarantees for large MRFs.



(a) (b) (c) (d) (e)

Figure 7.5: (a) 2 variable-disjoint cycles MRF of size 4 and (b-e) sets of variables covered by the SLT-region.

Example 7.3.2.3. *Consider the MRF composed of two variable-disjoint cycles of size 4 depicted in figure 7.5(a). To create the region, we remove each of the variables of the first cycle, one at a time (see figures 7.5(b)-(e)). We act analogously with the second cycle. Hence, $\mathcal{C}$ is composed of 8 elements. Just by counting we observe that each edge is completely covered 6 times, so $cc_* = 6$. Since we are removing a single variable at a time, $nc_* = 0$. Hence, the bound for a Max-Sum solution in this MRF structure is $6/8 = 3/4$.*

The following result generalizes the previous example to MRFs containing d variable-disjoint cycles of size larger than or equal to l.

Proposition 16. *For any MRF such that every pair of cycles is variable-disjoint and where there are at most d cycles of size l or larger, and for any Max-Sum solution x^{MS}, we have that:*

$$\theta(x^{MS}) \geq \left(1 - \frac{2(d-1)}{d \cdot l}\right) \cdot \theta(x^*) = \frac{(l-2) \cdot d + 2}{l \cdot d} \cdot \theta(x^*). \tag{7.9}$$

For the sake of readability, the proof is omitted here but can be consulted in Appendix C. The proof generalizes the region explained in example 7.3.2.3 to any variable-disjoint cycle MRF by defining a region that includes an element for every possible edge removal from every cycle but one.

Equation 7.9 shows that the bound: (i) decreases with the number of cycles; and (ii) increases as the maximum number of variables in each cycle grows. Figure 7.2(b) illustrates the relationship between the bound, the number of cycles (d), and the maximum size of the cycles (l). The first thing we observe is that the size of the cycles has more impact on the bound than the number of cycles. In fact, observe that by equation 7.9, the bound for a variable-disjoint cycle graph with a maximum cycle size of l is at least $\frac{(l-2)}{l}$, independently of the number of cycles. Thus, if the minimum size of a cycle is 20, the quality for a fixed point is guaranteed to be at least 90%. Hence, quality guarantees for Max-Sum solutions are good whenever: (i) the cycles in the MRF do not share any variables; and (ii) the smallest cycle in the MRF is large. Therefore, our result confirms and refines the recent results obtained for single-cycle MRFs (Weiss, 2000).

7.4 Conclusions

Although Max-Sum comes with no quality guarantees (neither from a designer not from a agent perspective), it is one of the leading incomplete DCOP algorithms due to

its good empirical performance. This chapter overcomes this limitation by proving that region optimality is a valuable tool to bound the quality of Max-Sum solutions.

Thus, the region optimal framework allows us, for the first time, to assess worst-case bounds on the quality of Max-Sum solutions for any problem (independent-problem guarantees) and for arbitrary graph structures (per-structure guarantees). On the one hand, we have proven that problem-independent guarantees rapidly decrease with the number of variables. On the other hand, we identified new classes of graph structures, besides acyclic and single-cycle, for which we can provide theoretical guarantees. As an example, we defined significant bounds for two-dimensional grids and graphs composed of variable-disjoint cycles. Concretely, we proved that:

- in two-dimensional grids Max-Sum solutions have at least 33% of the quality of the optimum;
- in large variable-disjoint cycles, such that smaller cycle contains at least 20 variables, Max-Sum solutions have at least 90% of the quality of the optimum.

Therefore, results presented in this chapter shed some light on the relationship between the quality of Max-Sum solutions and the graph structure of the model.

The quality guarantees we defined for Max-Sum are important for the DCOP community given the few DCOP algorithms, C-DALO and MGM-$\{2,3\}$ algorithms, that can actually provide system designer's quality guarantees. Moreover, since Max-Sum is widely employed to solve approximately the MAP problem in graphical models, our contribution it is also of interest for many other research areas such as statistical physics, computer vision or error-correcting coding theory (refer to (Wainwright and Jordan, 2008) for a description of Max-Sum application in such areas).

Figure 7.1 shows the resultant DCOP landscape after incorporating the aforementioned quality assessment for the Max-Sum algorithm. Notice that now, Max-Sum is included in the category of algorithms that can provide system designer's quality guarantees, concretely region optimal quality guarantees.

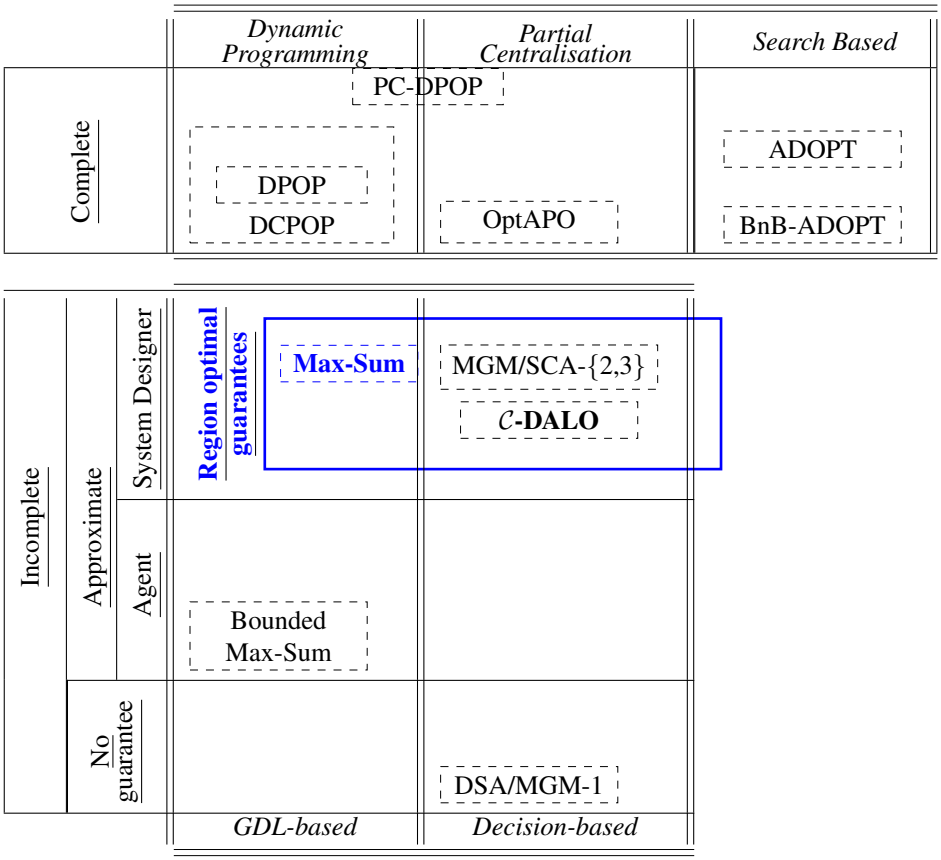


Table 7.1: DCOP algorithms landscape after Max-Sum quality assessment. Contributions of this chapter are highlighted in blue/bold. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

Chapter 8

Conclusions and Future work

In this chapter, we draw some conclusions about the work developed in this dissertation and we show some paths open to future development.

8.1 Conclusions

In this thesis, we have investigated different approaches to overcome the challenges that the problem of quality assessment under resource-boundedness poses on the design of DCOP algorithms. With this purpose, we focused on exploring various efficiency-related trade-offs along two dimensions: the level of required resources and the quality assessment over solutions.

On the one hand, under resource-boundedness, the trade-off to be considered was cost versus optimality. Some bounded optimization problems require complete algorithms that make the most effective use of resources to find an optimal coordination, while others require incomplete algorithms that can rapidly solve large-scale problems at the cost of finding suboptimal solutions. One of the main contributions of this thesis was to present algorithms in both categories:

(i) a complete DCOP algorithm: *Action-GDL*; and

(ii) three incomplete DCOP algorithms: *DaCSA* and *EU-DaC*, from the DaC family, and *C-DALO*, a generic region optimal algorithm.

On the other hand, quality assessment was identified as fundamental to enable a reasoned trade-off between the solution's quality and cost of incomplete algorithms and between the quality and the characteristics of the problem. These trade-offs required a broader concept of guarantees that include, in addition to optimality, approximate guarantees from two perspectives: from agents and from a system designer perspective. Thus, the second main contribution of this thesis consisted in providing two frameworks for approximate guarantees:

(i) *Divide-and-Coordinate*, which defines a family of algorithms with agent's quality guarantees; and

	Approximate Guarantees				
	Time		Specificity		
	Design	**Run**	**Problem-independent**	**Per-Class**	**Per-Instance**
ADOPT[1]	✓		✓		
BnB-ADOPT[1]	✓		✓		
A-DPOP	✓		✓		
DSA					
MGM-1					
Max-Sum	✓		✓	✓	
Bounded Max-Sum					✓
MGM/SCA-{2,3}	✓		✓	✓	
$\mathcal{C}$-DALO	✓		✓	✓	
DaCSA		✓			✓
EU-DaC		✓			✓

Agent's quality guarantees
System designer's quality guarantees

[1] ADOPT and BnB-ADOPT here refer to the bounded-error approximation extensions of the respective complete algorithms.

Table 8.1: Quality assessment landscape for incomplete DCOP algorithms after including this thesis contributions (shown in bold). Guarantees are characterised based on two dimensions: time at which they are available and specificity.

(ii) *Region Optimality*, which defines system designer's quality guarantees for a class of DCOP solutions.

As an additional benefit that stems from region optimality, and therefore contribution, we were able to assess system designer's quality guarantees for the Max-Sum solutions on convergence. Table 8.1 shows the algorithms for which we provided approximate quality guarantees based on these frameworks. As in chapter 2, quality guarantees are classified along two dimensions: the time at which they are available and their specificity. For DaCSA and EU-DaC we assessed agent's quality guarantees, namely runtime per-instance guarantees. In contrast, for $\mathcal{C}$-DALO and Max-Sum we assessed system designer's quality guarantees, namely design-time problem-independent and per-class guarantees.

All these contributions are founded on exploiting the structure of DCOPs, to design efficient DCOP algorithms that: (i) assess good solutions subject to the resources available; and (ii) bound the quality of these solutions.

Figure 8.2 presents a comparative overview of the current DCOP landscape after including this thesis contributions. Existing algorithms are shown together with the new algorithms developed in this thesis (the latter ones are shown in bold).

With these contributions we were able to overcome the main limitations of the DCOP literature when designing DCOP algorithms with quality guarantees. As dis-

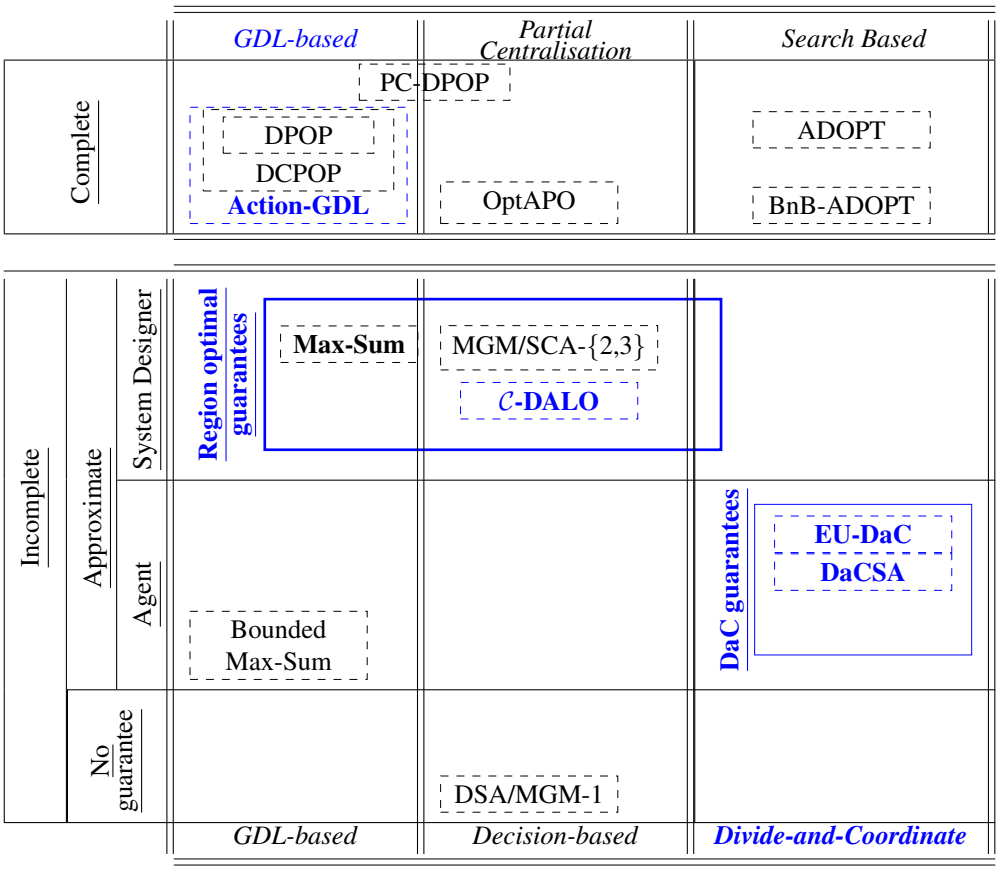


Table 8.2: DCOP algorithms landscape after including this thesis contributions (in bold). Contributions of this thesis are highlighted in bold/blue. DCOP algorithms are classified based on the quality assessment they provide over their solutions (vertical axis) and the approach they follow to solve DCOPs (upper and lower horizontal axes).

cussed at the end of chapter 3 (section 3.4), thus required to:

- explore efficient problem representations for optimal DCOP solving;
- design incomplete DCOP algorithms that assess agent's quality guarantees;
- extend the set of local optimal solutions that allow system designer's quality guarantees; and
- assess quality guarantees over Max-Sum solutions.

Below, we summarize in more detail the contributions of this thesis that fulfilled each of the four requirements listed above.

8.1.1 On exploring efficient problem representations for optimal DCOP solving

As discussed in chapter 1, the most challenging characteristic to endow complete DCOP algorithms with is efficiency. As reviewed in chapter 3, the efficiency of current state-of-the-art complete DCOP algorithms highly depends on the problem representation that, with exception of DCPOP (that exploits cross-edge trees), was limited to the space of pseudotrees. Against this background, in chapter 3 (section 3.4), we formulated some fundamental questions that must be explored in order to overcome the limitations of optimal DCOP approaches to exploit more general problem representations, namely:

- Is there any further problem representation that can be exploit by complete approaches? Can we theoretically or/and empirically characterize the potential improvement to explore such problem representation?

Next, we recapitulate how this dissertation, by means of the introduction of the Action-GDL algorithm in chapter 4, set the foundations to explore these fundamental research questions.

To formulate Action-GDL, we start from the GDL algorithm and extended it to solve DCOPs efficiently. Then, we showed that Action-GDL effectively reduces communication and computation with respect to GDL when solving DCOPs. Likewise GDL, Action-GDL required the problem to be compiled into a junction tree structure. Thus, we build on an existing distributed method for compiling junction trees in the literature (Paskin et al., 2005) to allow agents in Action-GDL to distributedly compile a DCOP into a junction tree.

In addition to the formulation of Action-GDL itself, we presented three sets of results that explore the generality and efficiency of the junction tree representation.

The first set of results characterised the generality of Action-GDL, and by extension of the junction tree representation. In particular, we provided two mappings that showed that junction trees extend the two representations currently used in optimal DCOP solving: (i) a mapping from pseudotrees (used by DPOP and search-based approaches) to junction trees; and (ii) a mapping from cross-edge trees (used by DCPOP) to junction trees. Then, based on these two mappings, we proved that Action-GDL generalises DPOP and DCPOP. By doing so we obtained a unifying theory for dynamic DCOP algorithms based on GDL. The reader can observe that the DCOP landscape in figure 8.2 now includes Action-GDL as a complete DCOP algorithm that subsumes DPOP and DCPOP.

The second set of results focused on characterising the efficiency of Action-GDL. On the one hand, we provided theoretical results that prove that moving from the space of pseudotrees (used by DPOP) to junction trees (used by Action-GDL) leads to significant improvements in terms of computation and communication. In particular, we observed that Action-GDL: (i) provides significant savings in computation over DPOP when pseudotrees are generated by edge-traversal heuristics; (ii) provides no significant savings in computation for unrestricted pseudotrees; and (iii) can severely reduce communication complexity.

On the other hand, we ran experiments to assess the improvement of the Action-GDL with respect to DCPOP. In order to obtain the maximum benefit from the junction

tree representation we formulated a novel distributed post-processing heuristic to optimize junction trees. Empirical results demonstrated that, by using this distributed post-processing heuristic, Action-GDL can be more efficient than DCPOP when the latter runs over the best cross-edge tree generated by state-of-the-art heuristics. These results confirmed the improved efficiency of Action-GDL with respect to current dynamic programming algorithms, now subsumed under the GDL framework.

Thirdly, we argued that several analytical benefits stemmed from the generality of the GDL framework. On the one hand, Action-GDL builds connections with a bunch of well-known algorithms used in other communities such as Viterbi's (Viterbi, 1967), Pearl's belief propagation (Pearl, 1988), or Shafer-Shenoy (Shafer and Shenoy, 1990) algorithms, to name a few, and with a wealth of theoretical results for GDL over junction trees (Aji and McEliece, 2000). In particular, in this dissertation we observed that this is also the case for the Cluster Tree Elimination algorithm (Dechter, 2003). On the other hand, Action-GDL builds a bridge between dynamic programming DCOP algorithms, namely DCOP and DCPOP, and the incomplete DCOP algorithms that also belong to the GDL framework, namely Max-Sum and bounded Max-Sum. Notice that, as depicted in figure 8.2, DPOP, DCPOP, Max-Sum and Bounded Max-Sum algorithms are unified under the same GDL-approach.

To summarise, the contributions in chapter 4 help overcome the current limitations of complete DCOP algorithms to exploit more efficient problem representations are:

- **Action-GDL**, a complete GDL-based DCOP algorithm that exploits a junction tree representation of the problem. Action-GDL builds upon:
  - **Extending GDL to solve DCOPs efficiently**. Action-GDL is defined as an extension to GDL to solve DCOPs efficiently, reducing communication and computation.
  - **Distributed compilation of junction trees.** We show agents can distributedly compile a DCOP into a distributed junction tree.
- **A mapping from the space of pseudotrees to junction trees.**
- **A mapping from the space of cross-edge trees to junction trees.**
- **A unifying theory for existing dynamic programming DCOP algorithms.** We showed that Action-GDL unifies the existing dynamic programming DCOP algorithms under GDL by generalising DPOP and DCPOP.
- **A distributed post-processing heuristic to optimize junction trees.**
- **Empirical and theoretical results that characterise the improvement in efficiency** of Action-GDL with respect to DPOP and DCPOP.

8.1.2 On assessing agent's quality guarantees

The second main contribution of this thesis was providing the means of assessing quality guarantees that can be used by agents at runtime. As identified in chapter 3 (section

3.4), with exception of the bounded Max-Sum algorithm, state-of-the-art DCOP algorithms fail to satisfy requirements for agent's quality guarantees, namely to be assessed at runtime and over the particular problem instance (per-instance quality guarantees). To overcome this limitation, we proposed a new family of DCOP incomplete algorithms, the so-called *Divide-and-Coordinate* (DaC) family, which enlarges this limited selection of algorithms.

The idea we explored in the DaC framework was to tackle the complexity of the problem by dividing it into simpler subproblems that are individually solved by each agent. We formulated the foundations of the DaC framework by means of two important properties, namely:

- The sum of the values of agents' local solutions, that we denoted as *value of a division*, bounds the quality of the optimal DCOP solution. We showed how agents can use this upper bound to return per-instance quality guarantees for their anytime solutions, defining in that way the DaC quality guarantees.

- The agreement among subproblems' local solutions stands for optimality. We formally proved that if all agents reach an agreement on a joint solution when optimizing their local subproblems, namely they assign the very same value to each variable in the DCOP, such solution is the optimal one.

The DaC approach founds on these two properties to solve a DCOP by searching for a division into subproblems such that subproblems' local solutions agree on their assignments. Thus, intuitively, the DaC approach aims to solve DCOPs by exploiting the concept of agreement.

Therefore, not only the DaC framework allows to assess per-instance quality guarantees that can be used by agents at runtime, the so-called DaC quality guarantees, but also defines a novel approach for DCOP solving. Figure 8.2 depicts the DCOP landscape including these two contributions.

To make the DaC approach operative we described how agents explore the space of divisions to find an agreement. To do that we defined an iterative process in which agents iterate through two stages: *divide* and *coordinate*. Recall that when agents solve their individual subproblems they may conflict by assigning different values to some shared variables. In order to explore the space of divisions efficiently, we proposed that agents exploit the information about their local solutions, along with their conflicts, when creating a new division. Thus, during each *divide* stage: each agent exchanges utilities with its neighbours to modify its local subproblem based on the exchanged information, and solves it to assess a new optimal local solution. During a *coordinate* stage agents exchange information about the conflicts on their local assignments.

The DaC framework allowed us as to define a novel family of incomplete DCOP algorithms by means of exploring three fundamental DaC dimensions: (i) the information agents exchange about their local subproblems; (ii) the local update of subproblems based on the information exchanged; and (iii) generation of candidate solutions close to an agreement as possible. Then, with the aim of exploring the space of DaC algorithms we: (i) formulated a generic DaC algorithm that realised the main operation of an incomplete DCOP algorithm when solving a DCOP by a DaC approach; and

(ii) formalised two DaC algorithms corresponding to two particular realisations of the above-mentioned unconstrained dimensions.

We started by formulating DaCSA, a DaC algorithm, in which agents exchange the most basic information: their local solutions. Then, as a strategy to reach an agreement, we proposed to update subproblems to favour neighbours' local solutions. To formalise DaCSA we employed well-known optimization techniques such as Lagrangian Dual Decomposition and subgradient methods.

To improve the performance of DaCSA and further explore the space of DaC algorithms, we proposed a second DaC algorithm: EU-DaC. EU-DaC was motivated by the intuition that the more information agents exchange, the more they get divisions closer to an agreement. Based on this idea, agents explicitly communicate the utilities of their assignments for their shared variables instead of their local solutions. To update subproblems, each agent exchanges utilities to compensate the difference with the utilities of its neighbours regarding their shared variables.

The complexity analysis we provided for these DaC algorithms showed that they are low-cost incomplete DCOP algorithms because during each iteration each agent: (i) exchanges a linear number of messages of linear size; and (ii) performs a linear number of operations.

Figure 8.2 depicts the inclusion of EU-DaC and DaCSA into the DCOP landscape to overcome the limited availability of algorithms with guarantees from an agent perspective.

The DaCSA and EU-DaC algorithms were experimentally analyzed when solving DCOPs that exhibit strong dependencies between agents' actions. We observed that: (i) DaC algorithms lead to better solutions on average than state-of-the-art DCOP algorithms that do not provide quality guarantees over their solutions; (ii) DaC quality guarantees are tight, and hence meaningful for agents at run time. Empirical results also demonstrated that EU-DaC outperforms DaCSA, acting as a support for the hypothesis that coordinating by exchanging the utilities of solutions instead of optimal solutions leads to higher quality solutions. In addition, experiments emphasize the different nature of quality guarantees, showing that system designer's guarantees as those provided by k-optimal algorithms, like MGM-$\{2,3\}$, are much looser than DaC quality guarantees, and hence, less suitable to be used by agents at runtime. As to solution quality, both approaches succeed in exploiting the problem structure to assess solutions of similar quality.

To summarise, the contributions in this dissertation regarding the assessment of agent's quality guarantees:

- ***Divide-and-Coordinate***, a novel approach to solve DCOPs that allows to assess agent's quality guarantees.
- **A generic DaC algorithm**, a generic incomplete DCOP algorithm.
- **DaCSA**, the first DaC algorithm in which agents coordinate by exchanging their solutions.
- **EU-DaC**, a second DaC algorithm that improves the performance of DaCSA by allowing agents to communicate their solutions' utilities.

- **An empirical evaluation** of the efficiency of DaCSA and EU-DaC when solving DCOPs and the accuracy of their guarantees.

8.1.3 On extending the set of local optimal solutions that allow system designer's quality guarantees

In what follows we summarise our contributions towards a general framework that overcomes the current limitations on the characterisation of local optimal solutions that allow system designer's quality guarantees.

As we analysed in chapter 2 (section 3.4), system designer's quality guarantees in the literature were limited to those provided by k-size and t-distance optimality over local optima characterised by size and distance. Then, to progress in this line of work we identified some fundamental questions that must be explored, namely:

(i) Can we define analogous approximate quality guarantees for a larger set of local optima, namely optima defined over arbitrary criteria?

(ii) Does a better criterion (beyond size and distance) exist that offers better guarantees, faster algorithms or more fine-grained control of the trade-off quality versus cost?

We observe that the region optimality framework proposed in chapter 6 provided the foundations to explore these two fundamental research questions.

To achieve that purpose, we started from the existing definitions of k-size and t-distance optimal solutions and generalised them to the notion of region optimality. Intuitively, we observed that both k-size and t-distance optimal solutions stand for DCOP assignments whose value can not be improved by changing the decision of any group of agents, what we called *neighborhoods*, in a region. Then, k-size and t-distance optimality differ on the criterion used to characterise the neighborhoods that compose regions: k-size neighborhoods are characterised by size whereas t-distance neighborhoods are characterised by distance to a central agent. We noticed that the set of neighbourhoods that compose a region is of central importance to both frameworks because they determine the degree of dominance of the local optima, affecting in that way the definition of the corresponding quality guarantees. In this dissertation we generalised these two classes of local optima by means of region optimal solutions, namely any local optimal solution in regions composed of neighborhoods characterised by arbitrary criteria.

In addition to region optimality, we also defined guarantees on the solution quality of any region optimal solution. As argued in chapter 1, system designer's quality guarantees, in addition to being assessed at design time, need to be general enough to apply to any problem instance that agents can face when deployed at runtime. With this aim, we provided region optimal guarantees that exploit different degrees of knowledge about problems' structure, namely:

- *Problem-independent region optimal guarantees* that apply to any problem;
- *Per-structure region optimal guarantees* that exploit knowledge about the graph structure;

- *Per-reward region optimal guarantees* that exploit knowledge about the reward structure;

To compute these region optimal quality guarantees we defined two mechanisms that differ on their computational cost. The first mechanism directly searches the space of problems to find a problem where the quality of the region optimum with respect to the global optimum is minimized. The main drawback of this mechanism is that it requires to generate and solve a linear program (LP) with a number of constraints exponential to the number of variables in the largest neighbourhood in the region. Hence, the fine mechanism guarantees tightness at the cost of worsening computational tractability. The second mechanism, allows to assess coarse region optimal guarantees, in linear time, though guarantees are in general not tight.

Finally, we proved that region-optimality generalises and unifies k-size optimality (Pearce and Tambe, 2007) and t-distance optimality (Kiekintveld et al., 2010).

Figure 8.2 depicts how region optimality defines a new type of system designer's quality guarantees that unifies k-size and t-distance algorithms, namely MGM-$\{2,3\}$ and $\mathcal{C}$-DALO.

With the formalisation of region optimality, a new dimension came into play. Indeed, region optimality allows to explore the space of local optimality criteria (beyond size and distance), looking for those that lead to better solution qualities. With the purpose of effectively showing that such better criteria exist, we analyzed the regions generated by k-size and t-distance under different graph structures. From this analysis, we concluded that the k-size criterion generates a potentially large number of neighborhoods of limited size, and t-distance generates a limited number of potentially huge neighborhoods. With the aim of keeping under control the number and size of neighborhoods, we introduced a new criterion, the so-called size-bounded distance criterion, which generates regions including a bounded number of limited-size neighborhoods.

Finally, we addressed the algorithm design challenge in region optimality by proposing $\mathcal{C}$-DALO, extension of DALO (Kiekintveld et al., 2010), which allows to search for region optima characterised by arbitrary criteria. $\mathcal{C}$-DALO operates following a decision-based approach: agents optimize in parallel neighborhoods in the region until reaching stability, and therefore a region optimal solution.

Table 8.1 shows the inclusion of $\mathcal{C}$-DALO as an incomplete DCOP algorithm that can provide agent's quality guarantees, namely problem-independent and per-class quality guarantees assessed at design time. Figure 8.2 shows the inclusion of $\mathcal{C}$-DALO in the DCOP landscape as a decision-based incomplete DCOP algorithm with region optimal guarantees.

Experimental results over different problem structures demonstrated that the average-case performance of $\mathcal{C}$-DALO with size-bounded-distance regions leads to better solution qualities than when employing k-size or t-distance criteria.

To summarise, the contributions of this dissertation related to extend the set of local optimal solutions that allow system designer's guarantees are:

- **Region Optimality**, a general framework that extends the class of DCOP solutions for which we can provide system designer's quality guarantees. Region Optimality develops along two dimensions:

  - **Characterisation of region optimal solutions.**
  - **Region optimal quality guarantees,** including per-structure region optimal quality guarantees and per-reward quality guarantees.

- **The size-bounded distance criterion**, a novel criterion to characterise neighborhoods that outperforms size and distance optimality.

- **The $\mathcal{C}$-DALO algorithm**, an asynchronous region optimal algorithm that allows to search for region optimal solutions in regions characterised by arbitrary criteria.

8.1.4 Quality guarantees for the Max-Sum algorithm

As thoroughly discussed in chapter 3 (section 3.4), the main drawback of Max-Sum algorithm stems from its lack of quality guarantees. To overcome this limitation we proved quality guarantees for Max-Sum solutions building upon region optimality.

With this aim we employed the results provided by Weiss and Freeman in (Weiss and Freeman, 2001), which that stated that any Max-Sum solution on convergence is guaranteed to be neighbourhood maximum in a region composed of all subsets of variables whose vertex-induced subgraph contains at most one cycle, the so-called Single Loops and Trees (SLT) region.

Based on this characterisation of Max-Sum solutions as region optimal, we proceeded to assess region optimal bounds for the SLT region. First, we showed how to assess tight quality guarantees for the SLT region by means of the fine mechanism. However, the exponential size that the SLT region exhibits in many graphs makes the complexity of generating the LP intractable, hence limiting the applicability of this mechanism to small-size scenarios. Nevertheless, the fine mechanism has analytical benefits. For example, it allows us to check the tightness of guarantees in small instances. Therefore, to assess computationally tractable guarantees for Max-Sum we needed to restrict to the faster mechanism. By means of the coarse mechanism we assessed region optimal quality guarantees for the SLT region that apply to Max-Sum solutions: (i) in any problem (problem-independent guarantees); and (ii) in problems with specific graph structures (per-structure quality guarantees).

As to the problem-independent quality guarantees, we observed that the quality guaranteed for the Max-Sum solution decreases rapidly with the size of the problem. Moreover, we found that, when no knowledge about the graph structure is exploited, the SLT-region corresponds to the 3-size region, and hence that any Max-sum solution is guaranteed to be at least 3-size region optimal.

As to per-structure quality guarantees, we showed that for problems with specific graph structures, we can assess much tighter quality guarantees than the introduced problem-independent guarantees. In particular, we defined SLT-region quality guarantees for bipartite graphs, two-dimensional grids and variable-disjoint cycle graphs that are calculated in constant time. For each of these graph structures, we analysed the relationship between the quality guarantees and the graph structure. As a result of this analysis, we identified new classes of graph structures for which the quality of max-sum solutions is guaranteed to be close to the optimum.

Two major benefits, and therefore contributions, derive from the introduced quality assessment over Max-Sum solutions. One the one hand, the provided quality guarantees help to shed some light on the relationship between the quality of Max-Sum solutions and the structure of the problem. We observed that these results are not only important for the DCOP community, but also in many other areas such as statistical physics, computer vision or error-correcting coding theory in which Max-Sum is used as an approximate inference algorithm. On the other hand, the characterisation of Max-Sum as a region optimal algorithm emphasizes the algorithmic-independent characteristic of the region optimal framework. As a result, the DCOP landscape in figure 8.2 now includes Max-Sum as an algorithm for which we can assess region optimal guarantees. To summarise, our contributions related to the quality guarantees of the Max-Sum algorithm are:

- **Max-Sum as a regional optimal algorithm**. By virtue of the characterisation of any Max-Sum solution as an SLT-region optimum, we proved region optimality as a valuable tool to bound the quality of solutions to which converge the Max-Sum algorithm. Our results build on:
  - the characterisation of any Max-Sum solution as neighbourhood maximum in a specific region of the MRF, the Single Loops and Trees (SLT) region (Weiss and Freeman, 2001); and
  - the region optimal quality guarantees, formalised in this dissertation.
- **Problem-independent SLT-region optimal guarantees for Max-Sum.**
- **Per-structure SLT-region optimal guarantees for Max-Sum,** quality guarantees for particular graph structures such as bipartite graphs, two-dimensional grids and variable-disjoint cycle graph structures.

8.2 Future work

While this thesis has realised significant contributions on exploiting the structure of problems to assess and bound multi-agent coordination, it also opens several paths to future developments. Concretely, the following areas seem specially promising for future work:

Exploiting the potential of the junction tree representation

As to the use of junction trees, we have only started to exploit its potential.

We do believe that we can exploit further theoretical and pragmatical tools derived from using this representation along several dimensions. Firstly, since the efficiency of Action-GDL depends on the underlying junction tree, investigating the potential existing junction trees heuristics (Cano and Moral, 1994; Amir, 2001; Flores et al., 2003) in a distributed environment becomes an interesting strand of research. Secondly, since junction tree based approaches, like CTE and Action-GDL, can eventually exchange large messages, a parallel line of research consists in extending these algorithms to exploit, in addition to the graph structure, the reward structure. Particularly, it would be

interesting to develop algorithms that reduce the size of messages by filtering out unnecessary information, along the lines of the work in (Pujol et al., 2011). Thirdly, based on our claiming that Action-GDL generalises DPOP, we consider that the multiple extensions to DPOP (Petcu, 2007) (e.g H-DPOP or MB-DPOP) might be generalised in terms of Action-GDL. From the generalisation of these extensions, we can expect not only an improvement of their efficiency due to the junction tree representation but also theoretical benefits due to the connection with the GDL framework.

Recall that in this dissertation we showed that by exploiting a more general problem representation, Action-GDL generalises existing dynamic programming DCOP algorithms improving their efficiency. However, this limitation on exploring more general problem representations is also present in search algorithms, like ADOPT and its extensions. Hence, as argued in (Yeoh et al., 2010), an open research area is to study how different representations from pseudo-trees affect the efficiency of Adopt approaches. Along this line, we believe that the theoretical and empirical results provided in this thesis about the benefits of exploring junction tree in dynamic programming approaches can serve as a basis and guide for analogous studies regarding ADOPT approaches.

Enhancing the DaC framework

As to the DaC framework, future work includes to study some unexplored aspects of this framework to allow a broad applicability of the DaC algorithms. The most interesting extension we envisage to DaC algorithms is to offer configurable trade-offs quality versus computational cost. Recall that the two proposed DaC algorithms explore the space of DCOP divisions restricted to subproblems with a particular graph structure that makes them computationally tractable. Moreover, when an agent exchanges information to coordinate with a neighbour, this information is restricted to its variable and the neighbour's variable. Therefore a natural extension to these algorithms amounts to allowing agents: (i) to handle more complex subproblems (incrementing computation) and (ii) to exchange information about their disagreement on larger combinations of shared variables (incrementing communication) to improve the quality of the DaC solution. In that way, agents would be able to trade-off solution quality versus cost because solutions generated in larger subproblems and exchanging more information are expected to be better. This poses the interesting problem of how agents select which information to incorporate in their subproblems and in their coordination messages that can decrease most their disagreement and lead to better solution qualities. At that point we do believe we can take advantage of several related works in the literature that propose strategies to realise this decision problem that emergence on dynamic trade-offs. Concretely several results are given in (Mailler and Lesser, 2004) for OptAPO in which agents follow several heuristics to dynamically increase the size of their subproblems as the problem solving unfolds or in (Petcu et al., 2007) for PC-DPOP, and extension to DPOP, where clusters of high width are distributedly detected by agents and centralised.

Secondly, as to the design of new DaC algorithms, we have only partially explored the dimensions that characterise the family of DaC algorithms. One path along which more efficient DaC algorithms can be developed is by exploring novel strategies to update subproblems or different information about their disagreement. Furthermore, since the efficiency of DaC algorithms highly depends on how well agents exploit suprob-

lems to assess anytime (candidate) solutions, the design of strategies to generate these solutions becomes an interesting strand of research. Although in this dissertation we contributed with two strategies, the *majority rule* and *the conditioned majority rule*, we do strongly believe that further strategies may exist and improve the efficiency of such algorithms. As an example, we can take inspiration of the way in which agents take decisions in OptAPO (Mailler and Lesser, 2004) where priorities ensure that the agents with the most knowledge over variables gets to make the decisions over them.

On the design of new region optimal algorithms and criteria

The region optimality framework formulated in this dissertation opens new research opportunities to study the design of new local optimality criteria and of more efficient region optimal algorithms. On the one hand, recall that region optimality allow us to: (i) define region optimal quality guarantees whereas exploiting some characteristics of the problem; and (ii) assess the complexity of searching for any region optimal with $\mathcal{C}$-DALO algorithm. Hence, since a critical issue in region optimality is the choice of regions, more than proposing new particular local optimality criteria what is needed is the design of techniques that allow us to explore the space of regions in search for regions with limited complexity and high quality guarantees. On the other hand, another line of further research opened by this thesis is the design of more efficient region optimal algorithms, that for arbitrary or particular regions, outperform $\mathcal{C}$-DALO. This would allow agents to search in larger regions that lead to better local optima. In particular, in this dissertation we showed that it is the case of Max-Sum algorithm: an efficient region optimal algorithm that allow agents to search for region optima in a particular large region of the graph, the SLT region.

On the efficiency of region quality guarantees to predict and trade-off Max-Sum performance

As discussed with detail in chapter 7, the theoretical guarantees provided in this thesis can shed some light on the relationship between the structure of the problem and the quality of the Max-Sum solutions on convergence. However, extensive experiments are needed in order to analyse the effectiveness of these quality guarantees when characterising the performance or trading-off quality versus cost in Max-Sum.

On the one hand, we plan to perform experiments to determine if effectively problem structures with higher quality guarantees lead to solutions of better quality. Moreover, since quality guarantees provided over Max-Sum only exploit the graph structure of the problem, further work also include to exploit the reward structure to assess more accurate guarantees. For example, we can easily incorporate the reward-based quality guarantees formulated in the region optimality framework, to characterise, in addition to the graph structure, which reward structures lead to better Max-Sum performance.

On the other hand, since quality guarantees provided only apply to Max-Sum solutions on convergence, the pragmatism of our bounds to predict Max-Sum performance is also very tied to the open problem of the characterization of the sufficient conditions of Max-Sum convergence, for which nowadays there are few theoretical results (Wiberg, 1996; Horn, 1999; Bayati et al., 2008).

Finally, we strongly believe that the quality guarantees provided in this dissertation

can be employed to study the problem of how to find a good set of regions in the Generalised Belief Propagation (GBP) algorithm (Yedidia et al., 2000; Welling, 2004), a generalisation of the Max-Sum that allows to trade-off quality versus cost getting better performance at expenses of incrementing the algorithm' complexity. Concretely, we point out that the quality guarantees defined over Max-Sum for particular structures can be used to guide the clustering of variables in GBP to generate graph structures for which we can provide high quality guarantees.

On the design of algorithms with hybrid quality guarantees

Although this thesis provided with important contributions on approximate quality assessment in DCOPs, observe in table 8.1 that all algorithms proposed are designed to assess one type of quality guarantees: agent's guarantees or system designer's guarantees. Therefore, further research aims to analyse current approaches with the aim of designing algorithms that can provide both: quality guarantees for the system designer and quality guarantees for the agents at runtime.

Privacy aspects

Notice that we have not analysed privacy aspects of any of the algorithms proposed in this dissertation. This is not within the scope of this dissertation since we were motivated by domains, such as sensor networks or traffic control, in which distribution has reasons of parallelism, communication costs or robustness. However, this lack of privacy analysis can limit the applicability of our contributions to some domains, such as distributed meeting scheduling, where privacy is the main issue. In this context, we strongly believe that the generality and well-theoretical characterisation of the different contributions presented in this thesis would allow to take advantage of some related works on privacy analysis in the literature that can be easily be transfer to the proposed DCOP algorithms. For example, we plan to take advantage for Action-GDL of the privacy-versions formulated for GDL message passing algorithms (Kearns et al., 2008) and for the region optimality framework of the privacy-analysis provided for the particular criterion of k-size optimality (Greenstadt, 2009).

Appendix A

Action-GDL generality proofs

In this appendix we provide proofs for lemmas 1 and 2, formulated in section 4.4.1 (chapter 4). Prior to proving lemmas 1 and 2 we would like to demonstrate a lemma and make two observations that will pave the way for these proofs.

Lemma 8. *The definition of cliques appearing in equation 4.8 is equivalent to the following recursive definition:*

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_k \in Ch(x_i)} \mathcal{C}_k^{'} \setminus \{x_k\} \right] \tag{A.1}$$

Proof. We prove the lemma by induction on the depth of variable x_i. In the base case we consider a variable x_i whose depth is 1, a leaf in the pseudotree. Observe that in that case both equations, eq. 4.8 and eq. A.1, define the x_i clique as its directed related variables $DRV(x_i)$.

Induction Step: Take a variable, x_i, whose depth is n+1.

Then, by eq. A.1, x_i clique is defined as:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} \mathcal{C}_j^{'} \setminus \{x_j\} \right]$$

By induction hypothesis we rewrite children cliques $\mathcal{C'}_j$ using eq.4.8:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} [DRV(x_j) \cup IRV(x_j)] \setminus \{x_j\} \right]$$

After eliminating $\{x_j\}$ from directed and inherited related variables:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} AP(x_j) \cup IRV(x_j) \right]$$

Since $P(x_j)$ is already contained in $DRV(x_i)$:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} IRV(x_j) \cup PP(x_j) \right]$$

Finally, by definition of inherited related variables (eq. 4.10):

$$\mathcal{C}_i^{'} = DRV(x_i) \cup IRV(x_i)$$

Hence we have proved that both definitions, $\mathcal{C}_i$ (eq. 4.8) and $\mathcal{C}_i^{'}$ (eq. A.1) are equivalent. □ □

Observation 1. *Observe that in mapping γ, the variables that compose a separator s^{ip} between a node x_i and its parent node x_p contains all variables from clique $\mathcal{C}_i$ excluding x_i. Formally, by equation A.1:*

$$s^{ip} = \mathcal{C}_i \cap \mathcal{C}_p = \mathcal{C}_i \cap \left[DRV(x_p) \ \cup \ \left[\bigcup_{x_k \in Ch(x_p)} \mathcal{C}_k \setminus \{x_k\} \right] \right]$$

We can take $\mathcal{C}_i$ out of the union since x_i is a child of x_p:

$$s^{ip} = \mathcal{C}_i \cap \left[DRV(x_p) \ \cup \ \mathcal{C}_i \setminus \{x_i\} \ \cup \bigcup_{\substack{x_k \in Ch(x_p) \\ k \neq i}} \mathcal{C}_k \setminus \{x_k\} \right]$$

Obvioulsy $\mathcal{C}_i \setminus \{x_i\} \subseteq s^{ip}$. Since x_i it is not in $DRV(x_p)$ nor in any of the remaining children cliques:

$$s^{ip} = \mathcal{C}_i \setminus \{x_i\} \tag{A.2}$$

Observation 2. *Considering what is stated by eq. A.2, we can reformulate the variables that compose each clique $\mathcal{C}_i$ in a JT defined by mapping γ as the union of the directly related variables of x_i with the union of all variables in the separators between clique $\mathcal{C}_i$ and each of its children. Formally*

$$\mathcal{C}_i = DRV(x_i) \ \cup \ \left[\bigcup_{x_k \in Ch(x_i)} \mathcal{C}_k \setminus \{x_k\} \right] = DRV(x_i) \cup \left[\bigcup_{x_k \in Ch(x_i)} s^{ki} \right]$$

Notice variables from a separator s^{ki} can also be expressed as the domain of the utility message μ_{ki} sent through this separator. Hence, the equation above can be rewritten as:

$$\mathcal{C}_i = DRV(x_i) \cup [\bigcup_{x_k \in Ch(x_i)} Scope(\mu_{ki})] \tag{A.3}$$

Observation 3. *Observe that given a pseudotree PT, the initial knowledge for DPOP in any node ($\mathcal{K}_i^0$) is equivalent to the potential defined on clique C_i (namely ψ_i) by mapping γ.*

Observation 4. *From the definition of $\mathcal{K}_i^0 = r^i \otimes [\bigotimes_{x_j \in AP(x_i)} r^{ij}]$ we can conclude that $Scope(\mathcal{K}_i^0) = DRV(x_i)$.*

We are now ready to proceed with the proof of Lemma 1.

Lemma 1. *Given a DCOP Φ and a pseudotree PT, the computation performed and the messages exchanged during the utility phase of $DPOP(\Phi, PT)$ and Action-GDL($\gamma(\Phi, PT)$) are the same.*

Proof

Proof. We prove Lemma 1 by showing that it is true for every variable x_i. We do that by induction on d, the depth of variable x_i in the pseudotree PT.

Consider case $d = 1$ (x_i is a leaf). During a DPOP execution, x_i exchanges with its parent x_p the following utility message (see eq. 4.6):

$$\mu_{ip} = \bigoplus_{\setminus x_i} [\mathcal{K}_i^0 \otimes \bigotimes_{x_j \in Ch(x_i)} \mu_{ji}] = \bigoplus_{\setminus x_i} \mathcal{K}_i^0$$

where the second equality follows from x_i being a leaf.

During the Action-GDL execution, clique $\mathcal{C}_i$, related to variable x_i, exchanges with its clique parent $\mathcal{C}_p$, related to variable x_p, the following utility message (see eq. 4.2):

$$\mu_{ip} = \bigoplus_{s^{ip}} [\psi_i \otimes \bigotimes_{\mathcal{C}_j adj\ \mathcal{C}_i, j \neq p} \mu_{ji}] = \bigoplus_{s^{ip}} \psi_i = \bigoplus_{\mathcal{C}_i \setminus \{x_i\}} \psi_i$$

where the second equality stems from the fact that by definition of γ (point 4) $\mathcal{C}_i$ is a leaf and the last equality follows from replacing s^{ip} by the expression obtained in observation 1 (eq. A.2).

For both algorithms to exchange the same message when x_i is a leaf, two conditions must hold: (1) both messages must combine the same relations and (2) both messages must summarize over the same variables. Point 1 is straightforward from observation 3. Regarding point 2, on the one hand, the set of variables that the DPOP message summarizes on is $Scope(\mathcal{K}_i^0) \setminus \{x_i\}$. From observation 4 follows that the set of variables that DPOP message summarizes on is $DRV(x_i) \setminus \{x_i\}$. On the other hand, the set of variables that the Action-GDL message summarizes on is $\mathcal{C}_i \setminus \{x_i\}$. From eq. 4.8 we can conclude that since $\mathcal{C}_i$ is a leaf, $\mathcal{C}_i = DRV(x_i)$ because it inherits no related variables. Thus, the Action-GDL message also summarizes on $DRV(x_i) \setminus \{x_i\}$ and messages are equal in the base case ($d = 1$).

Assume that lemma 1 holds for all variables whose depth is less than or equal to n in the pseudotree. Now consider variable x_i whose depth is $n + 1$. We consider the subtree having x_i as root, with all variables under x_i in the pseudotree whose depth is at most n. Since by induction both algorithms exchange the same utility messages for

variables in these subtrees whose depth is equal to or less than n, all the utility messages sent by variables under x_i are equal in both executions. Hence, we only have to prove that lemma 1 holds for utility messages sent by x_i.

During the execution of DPOP, variable x_i exchanges with its parent x_p the following utility message (see eq. 4.6):

$$\mu_{ip} = \bigoplus_{\setminus x_i} [\mathcal{K}_i^0 \otimes \bigotimes_{x_j \in Ch(x_i)} \mu_{ji}]$$

During Action-GDL execution clique $\mathcal{C}_i$, related to variable x_i, exchanges with its parent clique $\mathcal{C}_p$, related to variable x_p, the following utility message (see eq. 4.2):

$$\mu_{ip} = \bigoplus_{s^{ip}} [\psi_i \bigotimes_{\mathcal{C}_j adj\ \mathcal{C}_i, j \neq p} \mu_{ji}] = \bigoplus_{\mathcal{C}_i \setminus \{x_i\}} [\psi_i \bigotimes_{\mathcal{C}_j adj\ \mathcal{C}_i, j \neq p} \mu_{ji}]$$

where the second equality comes from replacing s^{ip} by the definition of separator in eq. A.2.

Again, for messages exchanged in DPOP and Action-GDL executions to be equal two conditions must hold: (1) both messages must combine same relations; and (2) both messages must summarize over the same variables.

Point 1 is straightforward from observation 3 and the fact that by induction the utility messages exchanged with their children are equal in both algorithms.

Now we will prove point 2. Firstly, observe that the set of variables that the DPOP message summarizes on is composed of the union of the scope of $\mathcal{K}_i^0$ excluding x_i and the scope of the utility messages exchanged with its children. From observation 4 the set of variables that the DPOP message summarizes on is composed of $\{DRV(x_i) \cup \bigcup_{x_j \in Ch(x_i)} Scope(\mu_{ji})\} \setminus \{x_i\}$. On the other hand, the set of variables that the Action-GDL message summarizes on is composed of $\mathcal{C}_i$ excluding x_i. Recall from equation A.3 that $\mathcal{C}_i = DRV(x_i) \cup \bigcup_{x_k \in Ch(x_i)} Scope(\mu_{ki})$. Therefore, since by induction messages exchanged for variables under x_i are the same in both algorithms, messages exchanged for x_i are equal in both algorithms and lemma 1 holds. □ □

Lemma 2. *Given a DCOP Φ and a pseudotree PT the value assigned by each agent to its variable and the messages exchanged during the value propagation phase of $DPOP(\Phi, PT)$ and Action-GDL($\gamma(\Phi, PT)$) are the same.*

Proof

Proof. We prove lemma 2 by showing that in both algorithms: (1) value messages exchanged contain assignments for the same variables; and (2) agents infer the same variables and assign the same values.

Firstly, the proof of (1) is quite straightforward given lemma 1. In DPOP, x_i composes a value message for each of its children, x_j, with assignments for variables in the scope of the utility message received from x_j in the previous phase (line 18, algorithm 2). In Action-GDL each clique $\mathcal{C}_i$, related to variable x_i, exchanges with each of its children clique $\mathcal{C}_j$, related to variable x_j, a value message with assignments for variables in their separator s^{ij} (line 19, algorithm 1). Recall that the scope of the utility

message exchanged in the utility phase between cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ during Action-GDL execution is equal to its separator s^{ij} (line 9, algorithm 1). Therefore, since by lemma 1 the utility messages exchanged by both algorithms are equal, so are their scopes, and the value messages exchanged during the value phase contain assignments for the same variables.

Secondly, to prove (2) we have to show that the values of variables inferred at each node (d^*) are equal in both algorithms. We recall that in both algorithms the values of variables inferred by x_i are:

$$d^* = d_i^* \cup \sigma_{pi}$$

where σ_{pi} is the value message received from x_i parent and d_i^* is the assignment for variables that maximizes its local knowledge, namely $\mathcal{K}_i$ and $\widehat{\mathcal{K}}_i$ in DPOP and Action-GDL respectively. The knowledge of a clique $\mathcal{C}_i$, related to variable x_i, in Action-GDL is (see equation 4.3):

$$\widehat{\mathcal{K}}_i = \underset{\sigma_{pi}}{\bigtriangledown}[\psi_i \otimes \bigotimes_{\mathcal{C}_j \in Ch_i} \mu_{ji}],$$

and knowledge of x_i in DPOP is (see equation 4.7) :

$$\mathcal{K}_i = \underset{\sigma_{pi}}{\bigtriangledown}[\mathcal{K}_i^0 \otimes \bigotimes_{x_j \in Ch(x_j)} \mu_{ji}].$$

Thus, to prove that each x_i infers the same set of variables with the very same value we jointly prove that for every node x_i: (1) the messages received from x_i parent and (2) their local knowledge are equal in both algorithms and (3) every messages σ_{ij} sent from x_i to its children x_j children are the same in both algorithms. We do that by induction on l, the level of x_i in the pseudotree (where root has level 1).

Consider case $l = 1$ (x_i is the root, with no parent). Point 1 is trivially true since x_i has no parent. Point 2 is satisfied by inspection of the two local knowledge definitions because (i) by observation 3, we have that $\mathcal{K}_i^0 = \psi_i$; (ii) by lemma 1 the utility messages exchanged in the utility phase are equal; and (iii) $\sigma_{pi} = \emptyset$ because x_i is the root. Finally, point 3, follows from noticing that since the knowledge is the same for both algorithms, the values inferred are the same. Hence, since the scopes of the messages are the same and what is send is just a portion of the inferred values, messages sent to children coincide.

Assume that points (1), (2) and (3) hold for all variables whose level is smaller or equal to n in the pseudotree. Now consider variable x_i whose level is $n + 1$. Applying (3) to x_i parent we find that the value messages received by x_i from its parent, σ_{pi}, are equal in both algorithms. Then the proof follows exactly the same reasoning than the case $l = 1$, only introducing the value message received, which we have proven to be equal. Hence both algorithms assign the same value to each variable and exchange the same value messages.

□

Appendix B

Region Optimality proofs

In this appendix, we develop the proofs that appear in chapter 7 into further detail. We present the proof of proposition 9 (section 6.4.1), proposition 10 (section 6.4.2) and proposition 11 (section 6.4.3).

Proposition 9. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region and β the minimum fraction reward. If $x^{\mathcal{C}}$ is a $\mathcal{C}$ optimum, then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \left(\frac{cc_*}{|\mathcal{C}| - nc_*} + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \right) \mathcal{R}(x^*),$$

where $cc_* = \min_{r \in \mathcal{R}} cc(r, \mathcal{C})$, $nc_* = \min_{r \in \mathcal{R}} nc(r, \mathcal{C})$, $pc_* = \min_{r \in \mathcal{R}} pc(r, \mathcal{C})$, and x^* is the optimal assignment.

Proof. For every $C^\alpha \in \mathcal{C}$, consider an assignment x^α such that: $x_i^\alpha = x_i^{\mathcal{C}}$ if $x_i \notin C^\alpha$, and $x_i^\alpha = x_i^*$ if $x_i \in C^\alpha$. Since $x^{\mathcal{C}}$ is $\mathcal{C}$-optimal, for all $C^\alpha \in \mathcal{C}$, $\mathcal{R}(x^{\mathcal{C}}) \geq \mathcal{R}(x^\alpha)$ holds, and hence:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \frac{\sum_{C^\alpha \in \mathcal{C}} \left(\sum_{r \in T(C^\alpha)} r(x^*) + \sum_{r \in N(C^\alpha)} r(x^{\mathcal{C}}) + \sum_{r \in P(C^\alpha)} r(x^\alpha) \right)}{|\mathcal{C}|}. \tag{B.1}$$

Using β we can express the third term in equation B.1 in terms of $\mathcal{R}(x^*)$ considering that the knowledge for any relation $r \in \mathcal{R}$ satisfies $r(x^\alpha) \geq \beta \cdot r(x^*)$. Therefore, the following inequalities hold:

$$\sum_{C^\alpha \in \mathcal{C}} \sum_{r \in P(C^\alpha)} r(x^\alpha) \geq \sum_{r \in \mathcal{R}} pc(r, \mathcal{C}) \cdot \beta \cdot r(x^*) \geq pc_* \cdot \beta \cdot \mathcal{R}(x^*) \tag{B.2}$$

From the proof of the general bound of equation 6.4 in section 6.2.3 we know that the first and the second sets of relations of equation B.1 can be also expressed in terms of $\mathcal{R}(x^{\mathcal{C}})$ and $\mathcal{R}(x^*)$. Therefore, after substituting these results in equation B.1 and rearranging terms, we obtain equation 6.10. □

Proposition 10. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP,$\mathcal{C}$ a region and δ a region optimal bound independent of the rewards. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \frac{1}{U}\left((U - L) \cdot \delta + L\right) \cdot \mathcal{R}(x^*)$$

where $U = \sum_{r \in \mathcal{R}} u_r$, $L = \sum_{r \in \mathcal{R}} l_r$.

Proof. Let $\hat{\mathcal{R}}$ be a distribution defined as $\hat{\mathcal{R}}(x) = \mathcal{R}(x) - L = \sum_{r \in \mathcal{R}}(r(x) - l_r)$. Notice that the rewards of $\hat{\mathcal{R}}$ are non-negative because after subtracting the minimum of each non-negative relation of $\mathcal{R}$ we obtain new relations in which the minimum value is 0. Moreover, because the value of any assignment in $\hat{\mathcal{R}}$ is equal to the value in $\mathcal{R}$ plus a constant, any $\mathcal{C}$-optimal $x^{\mathcal{C}}$ in $\mathcal{R}$ is also $\mathcal{C}$-optimal in $\hat{\mathcal{R}}$. Thus, by definition of $\mathcal{C}$-optimal bounds the following inequality holds:

$$\hat{\mathcal{R}}(x^{\mathcal{C}}) \geq \delta \cdot \hat{\mathcal{R}}(x^*) \tag{B.3}$$

Then by expressing $\hat{\mathcal{R}}$ in terms of $\mathcal{R}$ and isolating $\mathcal{R}(x^{\mathcal{C}})$ we obtain:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \delta \cdot \mathcal{R}(x^*) + (1 - \delta) \cdot L \tag{B.4}$$

Now multiplying and dividing the right equation side by $\mathcal{R}(x^*)$:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \left(\frac{\delta \cdot \mathcal{R}(x^*) + (1 - \delta) \cdot L}{\mathcal{R}(x^*)}\right) \cdot \mathcal{R}(x^*) \tag{B.5}$$

Since the bound provided by equation B.5 above increases as the value of the optimum, $\mathcal{R}(x^*)$, decreases, we can get rid of $\mathcal{R}(x^*)$, which is in general unknown, by replacing it with an upper bound. By definition, U is an upper bound of $\mathcal{R}(x^*)$. Hence, we can substitute U for $\mathcal{R}(x^*)$ to obtain equation 6.11. □

Proposition 11. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region, β the minimum fraction reward and $\delta = \frac{cc_*}{|\mathcal{C}| - nc_*}$, then*

$$\delta + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \leq \frac{1}{U}\left((U - L) \cdot \delta + L\right)$$

where $U = \sum_{r \in \mathcal{R}} u_r$, $L = \sum_{r \in \mathcal{R}} l_r$.

Proof. After rearranging terms and simplifying, we obtain that equation 6.12 is equivalent to:

$$pc_* \cdot \beta \leq (|\mathcal{C}| - nc_* - cc_*) \cdot \frac{L}{U} \tag{B.6}$$

First, from the definition of partial covering over any relation $r \in \mathcal{R}$, $pc(r, C) = |\mathcal{C}| - nc(r, C) - cc(r, C)$, we observe that $pc(r, C)$ increases as $nc(r, C)$ and $cc(r, C)$ decrease. Since nc_*, cc_* are the minimum values that functions nc, cc can take on respectively, then $pc_* \leq |\mathcal{C}| - nc_* - cc_*$ holds. Therefore, proving that $\beta \leq \frac{L}{U}$,

namely that $min_{r \in \mathcal{R}} \frac{l_r}{u_r} \leq \frac{\sum_{r \in \mathcal{R}} l_r}{\sum_{r \in \mathcal{R}} u_r}$, is enough to prove that equation B.6 holds. We build the proof by induction of the number of relations $n = |\mathcal{X}|$. Consider without loss of generality a problem with n relations such that $\frac{l_{r_{V_1}}}{u_{r_{V_1}}} \leq \ldots \leq \frac{l_{r_{V_{n-1}}}}{u_{r_{V_{n-1}}}} \leq \frac{l_{r_{V_n}}}{u_{r_{V_n}}}$ holds. If $n = 2$, then $min(\frac{l_{r_{V_1}}}{u_{r_{V_1}}}, \frac{l_{r_{V_2}}}{u_{r_{V_2}}}) \leq \frac{l_{r_{V_1}} + l_{r_{V_2}}}{u_{r_{V_1}} + u_{r_{V_2}}}$ simplifies to $\frac{l_{r_{V_1}}}{u_{r_{V_1}}} \leq \frac{l_{r_{V_2}}}{u_{r_{V_2}}}$, which by problem definition is true. When $n > 2$ we must prove that $\frac{l_{r_{V_1}}}{u_{r_{V_1}}} \leq \frac{l_{r_{V_1}} + \sum_{j \geq 2} l_{r_{V_j}}}{u_{r_{V_1}} + \sum_{j \geq 2} u_{r_{V_j}}}$ holds, or equivalently that $\frac{l_{r_{V_1}}}{u_{r_{V_1}}} \leq \frac{\sum_{j \geq 2} l_{r_{V_j}}}{\sum_{j \geq 2} u_{r_{V_j}}}$. By recursively applying the expression for $n = 2$ to $\frac{\sum_{j \geq 2} l_{r_{V_j}}}{\sum_{j \geq 2} u_{r_{V_j}}}$ we obtain that: $\frac{\sum_{j \geq 2} l_{r_{V_j}}}{\sum_{j \geq 2} u_{r_{V_j}}} = \frac{l_{r_{V_2}}}{u_{r_{V_2}}} + \frac{\sum_{j \geq 3} l_{r_{V_j}}}{\sum_{j \geq 3} u_{r_{V_j}}} \geq \min(\frac{l_{r_{V_2}}}{u_{r_{V_2}}}, \frac{\sum_{j \geq 3} l_{r_{V_j}}}{\sum_{j \geq 3} u_{r_{V_j}}}) \geq \ldots \geq \min(\frac{l_{r_{V_2}}}{u_{r_{V_2}}}, \min(\frac{l_{r_{V_3}}}{u_{r_{V_3}}}, \ldots, \min(\frac{l_{r_{V_{n-1}}}}{u_{r_{V_{n-1}}}}, min(\frac{l_{r_{V_n}}}{u_{r_{V_n}}}) \ldots) = \frac{l_{r_{V_2}}}{u_{r_{V_2}}}$. Thus, $\frac{\sum_{j \geq 2} l_{r_{V_j}}}{\sum_{j \geq 2} u_{r_{V_j}}} \geq \frac{l_{r_{V_2}}}{u_{r_{V_2}}}$ and consequently, $\frac{\sum_{j \geq 2} l_{r_{V_j}}}{\sum_{j \geq 2} u_{r_{V_j}}} \geq \frac{l_{r_{V_1}}}{u_{r_{V_1}}}$ holds. □

Appendix C

Max-Sum bounds proofs

In this appendix, we develop the proofs that appear in chapter 7 into further detail. We present the proof of proposition 12 (section 7.3.1), proposition 14 (section 7.3.2), proposition 15 (section 7.3.2), and the proposition 16 (section 7.3.2).

Proposition 12. *Let $\mathcal{G} = \langle \mathcal{X}, E \rangle$ be a graphical model and $\mathcal{C}$ the SLT-region in $\mathcal{G}$. Let $\mathcal{G}' = \langle \mathcal{X}', E' \rangle$ be a subgraph of $\mathcal{G}$. Then the bound of equation 6.4 for $\mathcal{G}$ holds for any SLT-optimal in $\mathcal{G}'$.*

Proof. We can compose a region $\mathcal{C}'$ containing the same elements as $\mathcal{C}$ just removing from each element of the region those variables that are not contained in $\mathcal{X}'$. Obviously $|\mathcal{C}| = |\mathcal{C}'|$.

Since we have to deal simultaneously with two regions in the proof, we use $cc_*^{\mathcal{C}} = \min_{(i,j)\in E} cc(\{x_i, x_j\}, \mathcal{C})$ and $nc_*^{\mathcal{C}} = \min_{(i,j)\in E} nc(\{x_i, x_j\}, \mathcal{C})$ when referring to $\mathcal{C}$. Likewise, we use $cc_*^{\mathcal{C}'} = \min_{(i,j)\in E'} cc(\{x_i, x_j\}, \mathcal{C}')$ and $nc_*^{\mathcal{C}'} = \min_{(i,j)\in E'} nc(\{x_i, x_j\}, \mathcal{C}')$ for $\mathcal{C}'$.

It is easy to see that $cc_*^{\mathcal{C}} = \min_{(i,j)\in E} cc(\{x_i, x_j\}, \mathcal{C}) \leq \min_{(i,j)\in E'} cc(\{x_i, x_j\}, \mathcal{C}) = \min_{(i,j)\in E'} cc(\{x_i, x_j\}, \mathcal{C}') = cc_*^{\mathcal{C}'}$ and that $nc_*^{\mathcal{C}} = \min_{(i,j)\in E} nc(\{x_i, x_j\}, \mathcal{C}) \leq \min_{(i,j)\in E'} nc(\{x_i, x_j\}, \mathcal{C}) = \min_{(i,j)\in E'} cc(\{x_i, x_j\}, \mathcal{C}') = nc_*^{\mathcal{C}'}$. Hence, the bound obtained applying equation 6.4 to $\mathcal{C}'$ is $\frac{cc_*^{\mathcal{C}'}}{|\mathcal{C}'|-nc_*^{\mathcal{C}'}} \geq \frac{cc_*^{\mathcal{C}}}{|\mathcal{C}'|-nc_*^{\mathcal{C}'}} = \frac{cc_*^{\mathcal{C}}}{|\mathcal{C}|-nc_*^{\mathcal{C}'}} \geq \frac{cc_*^{\mathcal{C}}}{|\mathcal{C}|-nc_*^{\mathcal{C}}}$. That is, we can obtain a bound for $\mathcal{G}'$ greater or equal than the bound obtained applying equation 6.4 to $\mathcal{C}$, as we wanted to prove. □

Proposition 14. *For any MRF with a n-m bipartite structure where $m \geq n$, and for any Max-Sum solution x^{MS} we have that*

$$\theta(x^{MS}) \geq b(n,m) \cdot \theta(x^*) \qquad b(n,m) = \begin{cases} \frac{1}{n} & m \geq n+3 \\ \frac{2}{n+m-2} & m < n+3 \end{cases} \tag{C.1}$$

Proof. Let $\mathcal{C}^A$ be a region including one out of the n variables and all of the m variables (in figure 7.3, elements (n)-(p)). Since the elements of this region are trees, we can guarantee optimality on them. The number of elements of the region is $|\mathcal{C}^A| = n$. It is clear that each edge in the graph is completely covered by one of the elements of $\mathcal{C}^A$, and hence $cc_* = 1$. Furthermore, every edge is partially covered, since all of the m variables are present in every element, and hence $nc_* = 0$. Applying equation 6.4 gives the bound $\frac{1}{n}$.

Alternatively, we can define a region $\mathcal{C}^B$ formed by taking sets of four variables, two from each side. Since the elements of $\mathcal{C}^B$ are single-cycle graphs (in figure 7.3, elements (b)-(j)), we can guarantee optimality on them.

Now, in order to apply proposition 6 to the region $\mathcal{C}^B$ we only need to assess $|\mathcal{C}^B|$, cc_* and nc_*. In a bipartite graph, there are $|\mathcal{C}^B| = \binom{n}{2} \cdot \binom{m}{2}$ different combinations of four vertexes when taking two from each side. Hence, the number of elements of $\mathcal{C}^B$ that completely cover any $(i,j) \in E$ is assessed as $cc(\{x_i, x_j\}, \mathcal{C}^B) = \binom{n-1}{1}\binom{m-1}{1} = (n-1)(m-1)$, because once fixed the two variables in $\{x_i, x_j\}$ we have to select an additional variable from each side but excluding the variable in $\{x_i, x_j\}$ already included. Clearly, $cc_* = (n-1)(m-1)$ as well. The number of elements of $\mathcal{C}$ that do not cover any $(i,j) \in E$ at all can be assessed as $nc(\{x_i, x_j\}, \mathcal{C}^B) = \binom{n-1}{2}\binom{m-1}{2}$ because we have to select two elements from each side but excluding the variables in $\{x_i, x_j\}$, which means one variable in each side. Obviously, $nc_* = \binom{n-1}{2}\binom{m-1}{2}$ as well. Applying proposition 6, we get the bound $\frac{2}{n+m-2}$. Observe that $\frac{2}{n+m-2} > \frac{1}{n}$ when $m < n+3$, and so equation 7.7 holds. □

Proposition 15. *For any MRF with an n grid structure where n is an even number, for any Max-Sum solution x^{MS} we have that*

$$\theta(x^{MS}) \geq \frac{n}{3n-4} \cdot \theta(x^*) \tag{C.2}$$

Proof. Let $\Gamma(l)$ be a division of l indexes into tuples of two elements, such that each index appears exactly in one tuple and any two indexes in the same tuple are non-consecutive. If l is even, the set of tuples $\Gamma(l) = \{(i, \frac{l}{2}+i) | i = 1 \dots \frac{l}{2}\}$ generates such division. Let $\mathcal{C}$ be a region formed by taking the vertices that result from the combination of any pair of rows $(i,j) \in \Gamma(n)$ and any pair of columns $(k,l) \in \Gamma(n)$. Note that optimality is guaranteed in each $C_i \in \mathcal{C}$ because variables in two non-consecutive rows and two non-consecutive columns create a single-cycle graph. Because $\Gamma(n)$ contains $\frac{n}{2}$ tuples, $|\mathcal{C}| = \left(\frac{n}{2}\right)^2$.

In a grid, any $(i,j) \in E$ contains two variables that are either in a column or a row. Let's consider without loss of generality that S contains two variables in a row with index i (the following discussion applies for columns by exchanging rows and columns). First observe that each index i appears exactly in one tuple in the division $\Gamma(n)$ of rows, and each tuple is combined with the $n/2$ tuples of the division $\Gamma(n)$ of columns. Hence, the number of elements of $\mathcal{C}$ that completely cover any $(i,j) \in E$ when (i,j) is either a row or a column is $cc(\{x_i, x_j\}, \mathcal{C}) = n/2$. Clearly, $cc_* = n/2$ as well. Secondly, we assess the number of sets in $\mathcal{C}$ that do not cover (i,j) at all. For any $(i,j) \in E$, such that the two variables in (i,j) are in a row, each variable

in $\{x_i, x_j\}$ appears in a single column. Notice that the two columns in which each variable in $\{x_i, x_j\}$ appears can not be in the same tuple in a division $\Gamma(n)$ because they are consecutive. Thus, there are $|\Gamma(n)| - 1$ rows in a division that does not contain any variable in $\{x_i, x_j\}$ and $|\Gamma(n)| - 2$ columns in a division that do not contain any variable in $\{x_i, x_j\}$. Hence, $nc(\{x_i, x_j\}, \mathcal{C}) = (\frac{n}{2} - 1)(\frac{n}{2} - 2)$ if S contains two variables in a row. Following a similar reasoning, $nc(\{x_i, x_j\}, \mathcal{C}) = (\frac{n}{2} - 2)(\frac{n}{2} - 1)$ if S contains two variables in a column. Obviously, $nc_* = (\frac{n}{2} - 2)(\frac{m}{2} - 1)$. Using these values in equation 6.4 provides equation 7.8. □

Proposition 16. *For any MRF such that every pair of cycles is variable-disjoint and where there are at most d cycles of size l or larger, and for any Max-Sum solution x^{MS} we have that*

$$\theta(x^{MS}) \geq \left(1 - \frac{2(d-1)}{d \cdot l}\right) \cdot \theta(x^*) = \frac{(l-2) \cdot d + 2}{l \cdot d} \cdot \theta(x^*). \tag{C.3}$$

Proof. Let $T = \{t_1, \cdots, t_l\}$ be a set of variables such that its vertex induced subgraph is a cycle of $l \geq 3$ variables in our MRF. It is evident that by removing any variable from T we break the cycle. Hence, we can define $T_{-k} = T \setminus \{t_k\} = \{t_1, \ldots, t_{k-1}, t_{k+1}, \ldots, t_l\}$, and its induced subgraph is not a cycle. Let U be the variables in the MRF that are not involved in any cycle.

By hypothesis, our graph contains at most d cycles of size $l \leq 4$. We can name them $T^1, \ldots, T^d$. It is clear that if we construct a set including: every variable that is not in any cycle (that is, U), a single cycle (say T^i) and for every remaining cycle T^i all the variables in it but one (say $T^j_{-k_j}$), its induced graph will have SLT-optimality guaranteed. In general for each choice of cycle (i) and each choice of variable to remove in the remaining cycles (k_j) we can create a set

$$C^i_{k_1,\ldots,k_{i-1},k_{i+1},\ldots,k_d} = U \cup T^i \cup \bigcup_{\substack{j \neq i \\ 1 \leq j \leq d}} T^j_{-k_j}.$$

Now, we can define the following region, including each of these sets

$$\mathcal{C} = \{C^i_{k_1,\ldots,k_{i-1},k_{i+1},\ldots,k_d} \mid \forall i \in \{1, \ldots, d\} \ \forall j \in \{1, \ldots, i-1, i+1, \ldots, d\}$$
$$\forall k_j \in \{1, \ldots, l\}\}$$

The total number of elements in the region is $|\mathcal{C}| = d \cdot l^{d-1}$, since we have to select one out of d cycles and for each of the remaining $d - 1$ cycles we have to remove one out of the l variables.

To compute cc_* we have to split our edges into four different types: (i) the ones that have both variables in U (S^1), (ii) the ones that have one variable in a cycle and the other one in U (S^2), (iii) the ones where each variable belongs to a different cycle (S^3), and finally (iv) the ones with both variables in the same cycle (S^4). We will assess cc_* as the minimum of these four values.

Since the variables in U appear in every element of the region, for each S^1 vertex with both variables in U we have that $cc(S^1, \mathcal{C}) = d \cdot l^{d-1}$.

Assume S^2 is an edge with one variable in U and another in one of the cycles (we can fix T^1 without loss of generality). Then S^2 appears in every element of the region that contains T^1 (we have l^{d-1}). Furthermore, for the elements of the region that include T^i completely (with $i > 1$), we can select one out of the $l - 1$ variables in T^1 which are not in S^2 and for each of the $d - 2$ cycles different from T^1 and T^i we should select one out of the l variables to remove. So, for each i we have $(l - 1) \cdot l^{d-2}$ and since we have $d - 1$ possible selections for i, we can conclude that $cc(S^2, \mathcal{C}) = l^{d-1} + (d - 1)(l^{d-2}(l - 1))$.

Assume S^3 is an edge with one variable in a cycle (say T^1) and another variable in a different cycle (say T^2). Including T^1 into our set, we have $(l - 1)$ choices for variables to remove from T^2 and l choices for variables to remove from the remaining $d - 2$ cycles, for a total of $(l - 1)l^{d-2}$. The same happens if we include T^2. If we include one out of the $d - 2$ remaining cycles, we have $l - 1$ choices for T^1 and $l - 1$ choices for T^2 and l choices for each of the remaining $d - 3$ cycles. Hence, we have that $cc(S^3, \mathcal{C}) = (l - 1)l^{d-2} + (l - 1)l^{d-2} + (d - 2)(l - 1)^2 l^{d-3}$.

Finally, assume S^4 is an edge with both variables in the same cycle (say T^1). Then S^4 appears in every element of the region that contains T^1 (we have l^{d-1}). Furthermore, for the elements of the region that include T^i completely with $i > 1$, we can select one out of the $l - 2$ variables in T^1 which are not in S^4, and for each of the $d - 2$ cycles different from T^1 and T^i we should select one out of the l variables to remove. So, for each i we have $(l - 2) \cdot l^{d-2}$, and since we have $d - 1$ possible selections for i, we can conclude that $cc(S^1, \mathcal{C}) = l^{d-1} + (d - 1)(l^{d-2}(l - 2))$.

It is easy to see that $cc(S^1, \mathcal{C}) \geq cc(S^2, \mathcal{C}) \geq cc(S^3, \mathcal{C}) \geq cc(S^4, \mathcal{C})$, and hence $cc_* = cc(S^4, \mathcal{C}) = l^{d-1} + (d - 1)(l^{d-2}(l - 2))$.

Since every edge in U is completely covered by every element of the region, we have that $nc_* = 0$.

Using these values into equation 6.4 and operating provides the desired result. □

Bibliography

Aji, S., Horn, G., Mceliece, R., and Xu, M. (1998). Iterative min-sum decoding of tail-biting codes. In *Proceedings of IEEE Information Theory Workshop*, pages 68–69.

Aji, S. M. and McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343.

Ali, S. M., Koenig, S., and Tambe, M. (2005). Preprocessing techniques for accelerating the dcop algorithm adopt. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 1041–1048.

Amir, E. (2001). Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI'01)*, pages 7–15.

Atlas, J. and Decker, K. (2007). A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 741–784.

Barabasi, A. L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Barbosa, V. (1996). *An Introduction to Distributed Algorithms*. The MIT Press.

Baxter, R. (1982). *Exactly Solved Models in Statistical Mechanics*. Academic Press, London.

Bayati, M., Borgs, C., Chayes, J. T., and Zecchina, R. (2007). Belief-propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *CoRR*, abs/0709.1190.

Bayati, M., Shah, D., and Sharma, M. (2008). Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251.

Bertsekas, D. (2007). *Nonlinear Programming*. Athena Scientific.

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition.

Bollobas, B. (2001). *Random Graphs*. Cambridge Press.

Bowring, E., Pearce, J. P., Portway, C., Jain, M., and Tambe, M. (2008). On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 607–614.

Bowring, E., Tambe, M., and Yokoo, M. (2006). Multiply-constrained distributed constraint optimization. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1413–1420.

Cano, A. and Moral, S. (1994). Heuristic algorithms for the triangulation of graphs. In *In Proceedings of the 5th International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'94)*, pages 98–107.

Chechetka, A. and Sycara, K. P. (2005). A decentralized variable ordering method for distributed constraint optimization. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 1307–1308.

Chechetka, A. and Sycara, K. P. (2006). No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1427–1429.

Davin, J. and Modi, P. J. (2005). Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 1057–1063.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Farinelli, A., Rogers, A., and Jennings, N. (2009). Bounded approximate decentralised coordination using the max-sum algorithm. *Proceedings of IJCAI-09 Workshop on Distributed Constraint Reasoning (DCR)*.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 639–646.

Feldman, J., Wainwright, M. J., and Karger, D. R. (2005). Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51(3):954–972.

Finley, M. R., Karakura, A., and Nbogni, R. (1991). Survey of intelligent building concepts. *IEEE Communication Magazine*.

Fitzpatrick, S. and Meertens, L. (2002). Experiments on Dense Graphs with a Stochastic, Peer-to-Peer Colorer. In *AAAI-02 Workshop on Probabilistic Approaches in Search*, pages 24–28.

Flores, M. J., Gámez, J. A., and Olesen, K. G. (2003). Incremental compilation of bayesian networks. In *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence (UAI'03)*, pages 233–240.

Freuder, E. C. and Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 1985)*, pages 1076–1078.

Frey, B. J., Koetter, R., Jr., G. D. F., Kschischang, F. R., McEliece, R. J., and Spielman, D. A. (2001a). Introduction to the special issue on codes on graphs and iterative algorithms. *IEEE Transactions on Information Theory*, 47(2):493–497.

Frey, B. J., Koetter, R., and Petrovic, N. (2001b). Very loopy belief propagation for unwrapping phase images. In *Proceedings of the Neural Information Processing Systems (NIPS)*, pages 737–743.

Gaston, M. E. and DesJardins, M. (2005). Agent-organized networks for multi-agent production and exchange. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence (AAAI 2005)*, pages 77–82.

Gershman, A., Meisels, A., and Zivan, R. (2009). Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research (JAIR)*, 34:61–88.

Greenstadt, R. (2009). An overview of privacy improvements to k-optimal dcop algorithms. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 1279–1280.

Gutierrez, P. and Meseguer, P. (2010). Saving Redundant Messages in BnB-ADOPT. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1259–1260.

Hirayama, K. and Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *Proceedings of Principles and Practice of Constraint Programming (CP97)*, pages 222–236.

Horn, G. B. (1999). *Iterative decoding and pseudocodewords*. PhD thesis, Department of Electrical Engineering, California Institute of Technology, Pasadena, CA.

Huang, B. and Jebara, T. (2007). Loopy belief propagation for bipartite maximum weight b-matching. In Meila, M. and Shen, X., editors, *In Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*.

Jebara, T. (2009). Map estimation, message passing, and perfect graphs. In *Proceedings of the 21th Conference in Uncertainty in Artificial Intelligence (UAI'09)*, pages 258–267.

Jensen, F. V. and Jensen, F. (1994). Optimal junction trees. In *Proceedings of the 10th Conference in Uncertainty in Artificial Intelligence (UAI'94)*, pages 360–366.

Junges, R. and Bazzan, A. L. C. (2008). Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 599–606.

Kask, K., Dechter, R., Larrosa, J., and Dechter, A. (2005). Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193.

Katagishi, H. and Pearce, J. P. (2007). KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *Proceedings of the 9th of the Distributed Constraint Reasoning (DCR) Workshop*.

Katsutoshi Hirayama, Toshihiro Matsui, M. Y. (2009). Adaptive price update in distributed lagrangian relaxation protocol. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 1033–1040.

Kearns, M., Tan, J., and Wortman, J. (2008). Privacy-preserving belief propagation and sampling. In *Proceedings of the Neural Information Processing Systems (NIPS)*, pages 745–752.

Khanna, S., Sattar, A., Hansen, D., and Stantic, B. (2009). An efficient algorithm for solving dynamic complex dcop problems. In *Proceedings of International Agent Technology Conference (IAT'09)*, pages 339–346.

Kiekintveld, C., Yin, Z., Kumar, A., and Tambe, M. (2010). Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 133–140.

Kok, J. R. and Vlassis, N. A. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828.

Krause, A., Singh, A. P., and Guestrin, C. (2008). Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284.

Laut, T. and Faltings, B. (2009). E[DPOP]: Distributed Constraint Optimization under Stochastic Uncertainty using Collaborative Sampling. In *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, pages 87–101.

Macarthur, K., Farinelli, A., Ramchurn, S., and Jennings, N. (2010). Efficient, superstabilizing decentralised optimisation for dynamic task allocation environments. In *Third International Workshop on: Optimisation in Multi-Agent Systems (OptMas) at the Ninth Joint Conference on Autonomous and Multi-Agent Systems*, pages 25–32.

MacKay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge Univ Press.

Maheswaran, R. T., Pearce, J. P., and Tambe, M. (2004a). Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems (ISCA-PDCS)*, pages 432–439.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., and Varakantham, P. (2004b). Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 310–317.

Mailler, R. and Lesser, V. R. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445.

Mailler, R. and Lesser, V. R. (2006). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Researh (JAIR)*, 25:529–576.

Mark, N. (2003). The structure and function of networks. *Society for Industrial and Applied Mathematics (SIAM) Review*, 45:167–256.

Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Conference in Uncertainty in Artificial Intelligence (UAI'99)*, pages 467–475.

Newman, M. and Watts, D. (1999). Renormalization group analysis of the small-world network model. *Physics Letters A*, 263:341–346.

Park, H., Burke, J., and Srivastava, M. B. (2007). Design and implementation of a wireless sensor network for intelligent light control. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 370–379.

Paskin, M. A., Guestrin, C., and McFadden, J. (2005). A robust architecture for distributed inference in sensor networks. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 55–62.

Pearce, J., Tambe, M., and Maheswaran, R. (2008). Solving multiagent networks using distributed constraint optimization. *AI Magazine*, 29(3):47–62.

Pearce, J. P. and Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1446–1451.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Petcu, A. (2007). *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, EPFL, Lausanne.

Petcu, A. and Faltings, B. (2005a). Approximations in distributed optimization. In *Proceedings of Principles and Practice of Constraint Programming (CP05)*, pages 802–806.

Petcu, A. and Faltings, B. (2005b). A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 266–271.

Petcu, A. and Faltings, B. (2008). Distributed constraint optimization applications in power networks. *International Journal of Innovations in Energy Systems and Power (IJESP)*, 3(1).

Petcu, A., Faltings, B., and Mailler, R. (2007). PC-DPOP: a new partial centralization algorithm for distributed optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 167–172.

Petcu, A., Faltings, B., and Parkes, D. C. (2008). M-DPOP: Faithful Distributed Implementation of Efficient Social Choice Problems. *Journal of Artificial Intelligence Researh (JAIR)*, 32:705–755.

Pujol, J. M., Delgado, J., Sangüesa, R., and Flache, A. (2005). The role of clustering on the emergence of efficient social conventions. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 965–970.

Pujol, M., Cerquides, J., Meseguer, P., and Rodriguez-Aguilar, J. A. (2011). Communication-constrained dcops: Message approximation in gdl with function filtering. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*. To appear.

Ramchurn, S. D., Farinelli, A., Macarthur, K. S., and Jennings, N. R. (2010). Decentralized coordination in robocup rescue. *The Computer Journal*, 53(9):1447–1461.

Rogers, A., Corkill, D. D., and Jennings, N. R. (2009). Agent technologies for sensor networks. *IEEE Intelligent Systems*, 24(2):13–17.

Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759.

Sanghavi, S., Malioutov, D. M., and Willsky, A. S. (2007). Linear programming analysis of loopy belief propagation for weighted matching. In *Proceedings of the Neural Information Processing Systems (NIPS)*, pages 1273–1280.

Sanghavi, S., Shah, D., and Willsky, A. S. (2008). Message-passing for maximum weight independent set. *CoRR*, abs/0807.5091.

Santoro, N. (2006). *Design and Analysis of Distributed Algorithms (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience.

Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating tasks in extreme teams. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 727–734.

Schiex, T., Fargier, H., and Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 631–639.

Shafer, G. and Shenoy, P. P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–351.

Silaghi, M.-C. and Yokoo, M. (2006). Nogood based asynchronous distributed optimization (adopt ng). In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1389–1396.

Singhvi, V., Krause, A., Guestrin, C., Jr., J. H. G., and Matthews, H. S. (2005). Intelligent light control using sensor networks. In *The ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 218–229.

Stranders, R., Farinelli, A., Rogers, A., and Jennings, N. R. (2009a). Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 601–608.

Stranders, R., Farinelli, A., Rogers, A., and Jennings, N. R. (2009b). Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 299–304.

Stranders, R., Fave, F. M. D., Rogers, A., and Jennings, N. R. (2010). A decentralised coordination algorithm for mobile sensors. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 874–880.

Sultanik, E., Lass, R. N., and Regli, W. C. (2009). Dynamic configuration of agent organizations. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 305–311.

Tambe, M., Bowring, E., Jung, H., Kaminka, G. A., Maheswaran, R. T., Marecki, J., Modi, P. J., Nair, R., Okamoto, S., Pearce, J. P., Paruchuri, P., Pynadath, D. V., Scerri, P., Schurr, N., and Varakantham, P. (2005). Conflicts in teamwork: hybrids to the rescue. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 3–10.

Tappen, M. F. and Freeman, W. T. (2003). Comparison of Graph Cuts with Belief Propagation for Stereo, using identical MRF Parameters. In *International Conference on Computer Vision (ICCV)*, pages 900–907.

Taylor, M. E., Jain, M., Jin, Y., Yokoo, M., and Tambe, M. (2010). When should there be a "me" in "team"?: distributed multi-agent optimization under uncertainty. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 109–116.

Vidal, R., Rashid, S., Sharp, C. S., Shakernia, O., Kim, J., and Sastry, S. (2001). Pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 2948–2955.

Vinyals, M., Rodríguez-Aguilar, J. A., and Cerquides, J. (2010). A survey on sensor networks from a multiagent perspective. *The Computer Journal*. In press. Published on-line on February 2010. DOI:10.1093/comjnl/bxq018.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Voice, T., Stranders, R., Rogers, A., and Jennings, N. R. (2010). A hybrid continuous max-sum algorithm for decentralised coordination. In *ECAI*, pages 61–66.

Wainwright, M. J., Jaakkola, T., and Willsky, A. S. (2004). Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166.

Wainwright, M. J., Jaakkola, T., and Willsky, A. S. (2005). Map estimation via agreement on (hyper)trees: Message-passing and linear programming. *CoRR*, abs/cs/0508070.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305.

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41.

Weiss, Y. and Freeman, W. T. (2001). On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744.

Welling, M. (2004). On the choice of regions for generalized belief propagation. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI'04)*, pages 585–592.

Wiberg, N. (1996). *Codes and decoding on general graphs*. PhD thesis, Department of Electrical Engineering, University of Linkoping, Sweden.

Yanover, C. and Weiss, Y. (2002). Approximate inference and protein-folding. In *Advances in Neural Information Processing Systems*, pages 84–86. MIT Press.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2000). Generalized belief propagation. In *Proceedings of the Neural Information Processing Systems (NIPS)*, pages 689–695.

Yeoh, W., Felner, A., and Koenig, S. (2010). Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 38:85–133.

Yeoh, W., Sun, X., and Koenig, S. (2009). Trading off solution quality for faster computation in dcop search algorithms. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 354–360.

Yin, Z. (2008). USC dcop repository. http://teamcore.usc.edu/dcop. University of Southern California, Department of Computer Science.

Zhang, W., Wang, G., Xing, Z., and Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87.

Zivan, R. (2008). Anytime local search for distributed constraint optimization. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 1449–1452.